AD-A038 137 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
IMAGE PROCESSING SOFTWARE CONVERSION. COMPUTER PROGRAM DOCUMENT--ETC(U)
FEB 77 J C LIETZ, M J GILLOTTE, D R HOUSE F30602-76-C-0164
UNCLASSIFIED PAR-76-35-VOL-2 RADC-TR-77-51-VOL-2 NL

1 OF 3
AD A038137

(12)
NW

RADC-TR-77-51, Volume II (of two)
Final Technical Report
February 1977

IMAGE PROCESSING SOFTWARE CONVERSION
Computer Program Documentation and Computer Programming Manual

Pattern Analysis and Recognition Corporation

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

IMAGE PROCESSING SOFTWARE CONVERSION, VOL II

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: David J. Brazil

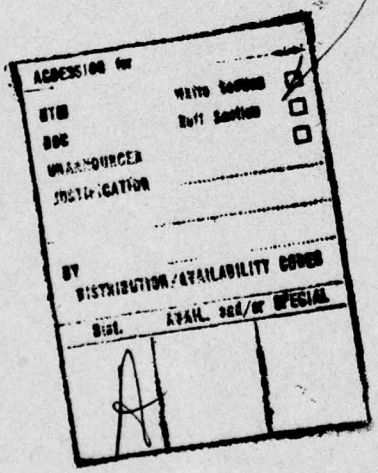
DAVID J. BRAZIL Capt, USAF
Project Engineer

APPROVED: Howard Davis

HOWARD DAVIS
Technical Director
Intelligence and Reconnaissance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office



Do not return this copy. Retain or destroy.

(18) RADC

(19) TR-77-51-Vol-2

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
RADC-TR-77-51, Volume II (of two)

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
(6) IMAGE PROCESSING SOFTWARE CONVERSION.
Computer Program Documentation and Computer Programming Manual.

5. TYPE OF REPORT & PERIOD COVERED
(9) Final Technical Report.
January [illegible] — December 1976,

6. PERFORMING ORG. REPORT NUMBER
PAR Report #76-35

7. AUTHOR(s)
(10) John C. Lietz,
Michael J. Gillotte
Donald R. House

8. CONTRACT OR GRANT NUMBER(s)
(15) F30602-76-C-0164

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Pattern Analysis and Recognition Corporation
228 On the Mall
Rome NY 13440

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
676244
62441074

11. CONTROLLING OFFICE NAME AND ADDRESS
(11) Rome Air Development Center (IRRE)
Griffiss AFB NY 13441

12. REPORT DATE
February 1977

13. NUMBER OF PAGES
12 218P.

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Same
(16) 6244
(12) 10

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.
(14) PAR-76-35-Vol-2

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Same

18. SUPPLEMENTARY NOTES
RADC Project Engineer: Captain David J. Brazil (IRRE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Computer Programming
Digital Image Processing
Disk Operating Systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This report describes the software design of the portion of the RADC image processing system that has been converted to execute on a PDP 11/45 computer. Certain changes in the design philosophy have been adopted in this conversion. The independent operating system approach has been dropped in favor of DEC's Disk Operating System. This allows a broader range of disk peripherals to be accessed at a reasonable system development cost. The data file I/O has been centralized to reduce future software development costs and to increase

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 390 101
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

reliability. In fact, several areas of the software have been modularized to reduce redundancy and to improve the maintainability of the overall software system.

The converted software package has also been improved from the viewpoint of man-machine interaction. Experience gained through use of the previous system has revealed certain areas where changes are desirable. One such improvement that was made was the adoption of a centralized error reporting scheme. This scheme allowed a more direct control over the errors that were reported by each routine. As a result, the errors are limited to a small set that explains the error conditions more clearly to the user. This and other improvements in the individual option capabilities have resulted in a more powerful and more easily used system.

TABLE OF CONTENTS

SECTION 1

SOFTWARE SYSTEM DESIGN

1.1. OVERALL SYSTEM ORGANIZATION

The image processing software is written to be executed on a PDP 11/45 as a program under the Disk Operating System (DOS). The software system exists as several separate program load modules and two files containing system tables. The executive control program and certain subroutines are contained within one file which is loaded into core by DOS at execution time. All other load modules consist of program functions related to the various options available to the user. The option lists which are presented to the user in frame format are stored in one file and the messages used for error reporting are stored in another.

The executive control program oversees the overall operations of the system. It accepts and interprets all frame level input (user entries when system is waiting for a frame option selection). The appropriate action is then taken to load and execute another program or to display a frame, whichever is indicated by the user selected option. Automatic recording of the user/system dialogue and the centralized error reporting function are also handled by the executive.

The executive calls upon the services of DOS to accomplish several functions. A DOS programmed request is used to load the option overlays into core in preparation for their execution. As these programs execute, they require access to data files stored on disk peripherals. This access is gained by requests to the executive which, in turn, uses DOS's programmed requests to access the files. Several other functions, such as stack maintenance and access to peripheral devices, are also accomplished via the services of DOS.

The system software can reside on any disk in the hardware configuration that is recognizable to DOS. However, it is typically stored with DOS on its system device. This places all software on one disk, leaving all others free for data storage. If there is only one disk in the system, it must be sufficiently large to accommodate DOS, the image processing software and all data files.

The data files are created within the DOS file structure and can therefore reside on any disk in the system. All software uses the DOS file specification (device, unit number, file name and extension) when referencing a file, and as a result, data file storage is not limited to one device as with the previous system. A program at any given time can access files which are resident on several different disk devices.

1.2. SOFTWARE ORGANIZATION

As previously mentioned, the software consists of an executive with related routines and several option overlays. The executive and each of the option overlays are stored in completely separate files. DOS is used to load and execute the executive module which then takes control. The executive accesses the frame file (FRAMES.IPS, ref. Appendix C) to initially display frame Ø and then to display all other frames as requested by the user. Each option in a frame describes the associated frame to be displayed or an option overlay to be executed.

1.2.1. Functions Of The System Frames

The frames are a key element in tying all system components together. In addition to providing the text to be listed on the graphics display for user option selections, each frame contains a table consisting of a list of three-word entries (ref. Appendix C). The first entry is the frame header which identifies the frame. All other entries correspond in order to the displayed options.

The frame is loaded into memory by the program "BLDISP", beginning at the global location "FRMTBL". The frame header and option entries reside in the first portion of this buffer. The frame text begins at "FRMLOC", which is the second part of the "FRMTBL" buffer. The text is packed in RAD50 format and therefore must be converted to ASCII prior to its display. After the text is displayed, this portion of the buffer is used for error reporting.

Following the display of the frame, the system waits for a user entry. If this entry is an option selection, the option number is converted to an index into the frame table currently residing in the "FRMTBL" buffer. This entry completely describes the action to be taken. New frame displays are indicated by the characters "FRM" packed into the first word of the entry in RAD50 format. The second word then contains the reference number of the new frame to be displayed.

If the selected option references an executable routine, the first word of the entry contains its three-character overlay reference numbers packed into RAD50 format. This is then combined with the RAD50 packed character "OVR" to form a six character name. A ".LDA" is then appended to complete the file name. This file contains the program to be loaded and executed.

As previously mentioned, each frame is identified by a reference number. Frames are retrieved from the disk file "FRAMES.IPS" by converting the number to an offset to the block in the frame file that contains the desired frame. The base address of the frame file is then added to this number to obtain the absolute disk address of the desired frame.

1.2.2. Overlay Loading

To avoid the unnecessary duplication of writing a program loader, the executive was written to use the services offered by DOS. The programmed request ".RUN" is used to load the option overlays into memory at a location immediately below the memory resident routines (ref. Appendix A). This function also moves the stack as required to allow the overlay to be loaded. The entry point for the overlay is then extracted from the selected frame option and control is transferred to the overlay via a subroutine call using general register five. One overlay may have several entry points. Each entry point is referenced by a number which is an offset in words from the first entry point. The first is designated in the normal fashion via the ".END" assembler directive and has an offset of Ø. The second entry point would then have a word offset of one.

After receiving control, the overlay then performs its programmed function using the services of the executive and the memory resident routines. These references to the executive subroutines are made possible by saving the symbol table which is generated when the executive module is linked. Each overlay is then linked with this symbol table to resolve the global references made by the overlay. Thus, complete linkage is attained.

A large buffer space is assigned within the executive program module. This buffer is intended to be used by the option overlays for computations and data file I/O. Access to this buffer is also gained via the symbol table. A parameter in the symbol table describes the size of the buffer which must be observed by each overlay. This allows the system to be built for differing amounts of memory available simply by varying the value of this parameter. The advantage gained by storing the buffer with the executive is that it is only stored on the disk once as opposed to once for each option overlay.

An option overlay can be broken down into suboverlays if necessary. These suboverlays, however, are loaded into core by an entirely different mechanism. The autoload software described in the "DOS/BATCH Linker (LINK) Programmer's Manual" is used to perform the suboverlay loading. In this scheme the option overlay is made up of a root section, which is memory resident throughout the execution of the option overlay, and several suboverlays. The suboverlay tree structure can be as complex as allowed by the DOS autoload software.

The autoload feature has several advantages. First, the option overlay and all of its suboverlays are stored in one file, thereby limiting time-consuming file searches to one. That one search locates the file and loads the root section into memory. The address of the file is then retained for future suboverlay loading.

A second advantage is the ease in which the option can be programmed. All overlay operations are transparent to the programmer, thus greatly simplifying his task.

One might inquire as to why the autoload feature was not used for the entire system. This reason becomes apparent when considering the requirements of the autoload software. Each overlay in the system must be described by a memory-resident autoload vector. There are currently fifty option overlays, several of which have suboverlays. Future system additions are expected to more than double this number. The memory requirement for autoload vectors would then become very large. By using the approach of two overlay loading schemes, the autoload vectors are no longer permanently resident in memory. They exist within the root section of their associated option overlays and are therefore not only reduced in number, but are in memory only for the duration of the option execution.

A second advantage is gained in the ease with which a new option can be added to the system. If only autoloading is used, the entire system would necessarily have to be relinked whenever a new option is added. By combining the two approaches, new options can be added by simply linking the new software with the executive symbol table. All previously existing software can be used without relinking.

It should be noted that some autoload vectors are permanently memory resident. This is due to the fact that file system routines are suboverlays of the executive and have been implemented within the autoload scheme. This provides rapid access to the file functions at a reduced memory requirement.

The ability to perform overlay autoloads from two separate files (executive and option overlay) simultaneously was achieved by replacing a portion of the DOS autoload software. The executive now performs the function of determining from which file the overlay is to be loaded. The standard DOS library routine for this function prohibits multiple file loading and therefore could not be used.

One last overlay option has been included. This feature allows one option overlay to request that the executive load another option overlay. When the second overlay is loaded, all traces of the first overlay are removed from memory and the stack is initialized (all parameters removed).

1.3. FILE SYSTEM

The image processing file system is designed within the DOS file structure. It is essentially an interface between the option overlays which access the files and the DOS file functions which maintain the files and their associated directories. Special considerations have been incorporated into this interface to achieve rapid data access and to provide high-level file functions tailored to image processing needs.

All files are created as contiguous files with the exception of the automatic log file (LOGFIL.IPS) which is linked to allow for expansion. The contiguous file format allows several logical blocks of data to be transferred between disk and memory in one access. This has the effect of greatly reducing data-transfer time.

The internal structure of the data files is organized into fixed length records. The record length, as well as the number of records, is dependent upon the type of data stored within. Each file begins with a 256-word header, in which are stored various parameters describing the data. Detailed descriptions of these parameters can be found in Appendix E.

The data records immediately follow the header beginning in the first available logical device block after the header. Records immediately follow one another in consecutive order with no intervening spaces. This, of course, means that for certain record sizes, one or more records may cross device block boundaries at one or more points in the file.

A file is created or retrieved by calling the appropriate file system routine with an accompanying parameter list called a "request block". The format of this block is as follows:

Word	Description
0	Address of a character string for requesting the file specification.
1, 2	File name
3	File name extension
4	Device
5	Unit number in low byte File access mode in high byte
6	Address of buffer for file operations
7	Size of buffer
8	Number of records in file
9	Record length
10	Number of consecutive records required in core simultaneously
11	Record number requested.

The request block describes the file to be created or retrieved and the buffer space to be used for data I/O. This buffer space must be dedicated to the file until it is closed, at which time it is available for other use.

If the first word of the request block contains an address of a character string, the string is printed at the keyboard display as a query for a device and a file name. The user entry is then placed in the request block. If the character string address is not present, the file specification appearing in the request block is used. In either case, the file is created or retrieved as specified. The number of records and record length must be provided by the calling program for create operations and are returned by the filing system for retrieve operations. The overlays request data records by placing the number of the desired record into word 11 of the request block. The file system routines then read this record into memory for the overlay. Due to the typically sequential access of image files during processing, the file system is designed to anticipate the next records that will be requested by the option overlays. Therefore, instead of reading just one record, the entire buffer specified in the request

block is filled with records that immediately follow the requested record. The result is that the next request does not require a disk access since the record has already been obtained in a previous access.

A similar procedure is followed when writing data onto a file. For this operation, the specified buffer is filled prior to transfer to its disk file. However, sequential file output is not only assumed in this case, it is expected. Due to the complexities involved in providing random access when records can cross logical block boundaries, this capability has not been included. For options that require such access, a read/write mode is available. In this mode the buffer contents are transferred to the file and then the records beginning with the requested record are read into memory. This eliminates the problem of preserving the overlapping record data.

Certain options require that several consecutive records be in memory simultaneously. This is accomplished by specifying the number of records in word 10 of the request block. The file system then insures that the requested block of records are available in memory. Since the buffer space required to load a block of records is computed when the file is opened, this information must be provided at that time. An error condition occurs if sufficient space is not available.

1.4. ERROR PROCESSING

A centralized error reporting scheme has been incorporated into the system. An error condition detected by any routine in the system is reported by a call to the error reporting routine with two parameters which specify the type of error and the message to be listed. Two types of errors are currently included. The first type is an error condition in which remedial action by the user allows the interrupted process to continue. Such errors are called recoverable errors. The second type of error condition results in a system state from which a recovery cannot be made. Such an error is termed a fatal error.

The errors are reported via a simple subroutine call to the error processing routine. The error type is specified by a number in the calling parameter list. A "Ø" indicates a recoverable error and a "1" denotes a fatal error. These correspond to two lists of messages that are maintained in a file under the name of "ERRORS.IPS". This file has a header block in which relative block numbers within the file are given for the start of each error list. The first action of the error processing routine then, is to determine the base disk address of the selected error list.

The second calling parameter indicates which message in the list is to be logged. The appropriate message is then read into memory and printed on the graphics display. If the error is recoverable, a normal subroutine return is then executed. If the error is fatal, control is transferred instead, to the executive which then terminates the execution of the option overlay and removes it from memory.

SECTION 2

PROGRAMMER'S MANUAL

This section is intended as a programming guide for developing new software for the image processing system. A variety of services are offered by the executive, the file system and several utility programs. A large number of these services can be conveniently referenced via a set of system macros (ref. Appendix H). Although in some cases the programmer may find that directly inserting the code is just as convenient as using a macro, this should be avoided. By consistent use of the macros, modifications or updates can be made to an entire system by simply changing a few macros. Without the use of macros, a time-consuming search must be made of all routines in the system for occurrences of the code sequences being changed.

In the remainder of this section, several programming conventions and callable subroutines are discussed. Sufficient information is provided in each case to properly use these subroutines. Further details may be found in the program documentation in Section 4. For this purpose, the name of the Program in which the subroutine is contained is given here.

2.1. SUBROUTINE CALLING CONVENTIONS

All subroutine calls follow the general format given below:

```
JSR  (reg.), (subroutine name)
BR   (offset to first instruction following the parameter list)
Parameter list
     .
     .
     .
```

This format has two advantages. First, by including the branch instruction, the offset (lower byte) of the branch gives the number of parameters. For routines that accept variable length lists, the number of parameters passed can readily be determined.

The second advantage is that DOS FORTRAN uses this calling format with register five in the subroutine call. Therefore, if the parameters are referenced in the list by their addresses, the routine is FORTRAN callable.

A general system macro has been provided to generate the proper subroutine calls. The format of this macro is as follows:

```
CALL      name,P1,P2,P3,P4,P5,P6
```

which expands to

```
JSR       R5,name
BR        (offset to first executable instruction)
```

```
.WORD     P1
.WORD     P2
.WORD     P3
.WORD     P4
.WORD     P5
.WORD     P6
```

The number of parameters is optional within the range of 0 to 6. If no parameters are specified, the branch statement is not generated.

The CALL macro should be used in all cases where a specialized macro is not available. In the subroutine descriptions to follow, the proper macro to be used in each case is discussed.

2.2. GRAPHICS DISPLAY INTERACTION

The graphics display is the prime means of communication with the system. This communication is accomplished via frame displays and user/system dialogue. Since this is a storage display, it must be erased and rebuilt from time to time to avoid overwriting of information. This rebuild action is under the control of the program "TELEIO" which together with "PLOT" and "TTYIO" contains the display I/O subroutines. Therefore, it is imperative that the user reference these subroutines for all display operations.

Since this is an interactive system, the programmer must frequently make requests of the user and accept his responses. These services are provided by the system at two levels. The higher-level routines are those which output a message and input the user's response. These consist of routines to request file specifications and numeric input. The lower level routines simply output or input character strings. These should not be used if a higher-level routine will satisfy the requirement.

Character strings consist of ASCII characters terminated by a null. Embedded carriage returns and line feeds are permitted. For normal user communications the line length should not exceed 40 characters. If a longer message is desirable, it should be sectioned into two or more lines with embedded carriage returns and line feeds. It should be noted that one complete message should not be output in sections by making several calls to the output routines. The initial lines of text may be lost if the display is rebuilt prior to completing the output. If the complete message is formatted with carriage returns and line feeds, then the output routines can insure sufficient display space for the entire message.

The available communication subroutines and their associated calling sequences are discussed below. The "CALL" macro is used in all cases to form the subroutine calling sequence.

2.2.1. Request File Specification

```
CALL    GETNAM,CHR,BUF
        (Ref. Program TTYIO)
```

where

CHR = Address of the output character string
BUF = Address of a 12-word buffer for return parameters

Return is via the branch instruction. Upon return, the twelve-word buffer contains, in order, a five-word link block and a seven-word file name block as defined for DOS. Device and unit default values (those used when none are entered by user) are provided by this routine. "Wild card" specifications as defined under DOS are returned unaltered (52_8 for any parameter indicates the wild card).

It should be pointed out that file names are normally requested via the file system routines (Para. 2.3). Only special circumstances require the use of the "GETNAM" routine.

2.2.2. Request a String of Signed Double Word Integer Numbers

To input decimal numbers:

```
CALL    DBLDEC,CHR,CNT,BUF,IND   (IND is optional)
        (Ref. Program TTYIO)
```

To input octal numbers:

```
CALL    DBLOCT,CHR,CNT,BUF,IND   (IND is optional)
        (Ref. Program TTYIO)
```

where

CHR = Address of the output character string

CNT = Address of a location containing the count of numbers to input. This count must be satisfied exactly or an error message is printed and a request is made to retype the line.

BUF = Address of buffer in which to return the numbers in the order entered. This buffer must contain sufficient space to store all the numbers indicated by the second parameter. Double word values are returned low order first followed by high order.

IND = (Optional parameter) Address of a location in which to return an indicator. A negative one is returned if a CTRL/Z is found to be the first character entered by the user. Otherwise, a zero is returned.

This routine is designed to allow one or more numbers to be entered on one or more lines by the user. For multiple line input where the total number of lines is determined by the user, the call to DBLDEC should be contained within a loop. This call should use the optional fourth parameter "IND". The loop is exited when IND = -1, which occurs when the user indicates the end of input by responding with a CTRL/Z.

2.2.3. Request a String of Signed Word Integer Numbers

To input decimal numbers:

```
CALL    SNGDEC,CHR,CNT,BUF,IND   (IND is optional)
        (Ref. Program TTYIO)
```

To input octal numbers:

```
CALL    SNGOCT,CHR,CNT,BUF,IND
        (Ref. Program TTYIO)
```

The parameters and their descriptions are identical to that given for DBLDEC except that word values are returned in the buffer.

2.2.4. Request a String of Signed Byte Integer Numbers

To input decimal numbers:

```
CALL    BYTDEC,CHR,CNT,BUF,IND   (IND is optional)
        (Ref. Program TTYIO)
```

To input octal numbers:

```
CALL    BYTOCT,CHR,CNT,BUF,IND   (IND is optional)
        (Ref. Program TTYIO)
```

The parameters and their descriptions are identical to that given for DBLDEC except that byte values are returned in the buffer.

2.2.5. Request a String of Floating Point Numbers

```
CALL    FLTENT,CHR,CNT,BUF,IND   (IND is optional)
```

The parameters and their descriptions are identical to DBLDEC except two-word floating point numbers are returned. Numbers are accepted in either of the following formats:

```
19678.66
1.967866E04
```

where both entries represent the same number.

2.2.6. Output a Character String

```
CALL    TTYOUT,CHR
        (Ref. Program TELEIO)
```

where

CHR = Address of the output character string. TTYOUT will append a final carriage return and line feed when the null terminator is encountered.

Following the call, the global location "TTYADR" contains the address of the byte immediately following the null in the output string. An additional output method is detailed below:

```
CALL    TTYO,CHR
        (Ref. Program TELEIO)
```

where

CHR = Address of the output character string. A final carriage return and line feed are not appended when the null terminator is encountered.

Following the call, the global location "TTYADR" contains the address of the byte immediately following the null in the output string.

2.2.7. Input a Character String

```
CALL    TTYIN,BUF
        (Ref. Program TELEIO)
```

where

BUF = Address of a buffer in which to deposit the string. This buffer should be of sufficient length to contain the expected string. The word immediately preceding the buffer should contain the size of the buffer in bytes. This number is used by TTYIN to prevent overflow. The terminating carriage return entered by the user is replaced with a null by TTYIN.

A general-purpose input buffer is provided within the executive for use with TTYIN. This buffer is defined by the global "TTYBUF". The size of this buffer (preceding word) is set to 80 bytes. Under no circumstances should the size of TTYBUF be altered.

2.2.8. Specialized Graphic Displays

Certain functions require that the graphics terminal be used for special display purposes. In this case, the display is temporarily under the control of the executing overlay. To gain control of the display, the following subroutine should be called:

```
CALL    DSABLR
        (Ref. Program TELEIO)
```

This disables the display rebuild software. The display can then be manipulated by using the graphic plot subroutines below in addition to any of the other terminal I/O subroutines. After the specialized display mode is no longer required, the following call should be made to enable the display rebuild software:

```
CALL    ENABLR
        (Ref. Program TELEIO)
```

The current frame is then redisplayed by the following call:

```
CALL    BLDISP
        (Ref. Program BLDISP)
```

Another special display feature is available to the programmer. This allows specialized displays to be presented and user/system dialogue to be maintained. To accomplish this, the address of a programmer-supplied routine that builds the specialized display must be placed in the global location "DSPADR". The following sequence of operations is then performed:

1. CALL DSABLR
2. Call the routine that presents the special graphics display. Space must be reserved along the left display margin for dialogue. This routine must leave the alpha cursor positioned at the top of the left margin.
3. CALL ENABLR

The dialogue can now be maintained with the system. When the dialogue reaches the bottom of the left margin, the control software will call the routine whose address appears at "DSPADR" to rebuild the display. Dialogue then continues.

Upon exiting the special display mode, the "BLDISP" routine should be called and the address "BLDISP" should be replaced in "DSPADR" to allow normal frame rebuild.

The following is a collection of routines to allow graphic plots to be constructed on the display terminal. The display manual should be consulted for complete descriptions of the various plotting operations. The calling sequences are as follows (all subroutines are in program PLOT):

To put the display in alpha mode:

```
CALL      ALPHA
```

To put the display in graphics mode:

```
CALL      GRMODE
```

To clear the display screen:

```
CALL      CLEAR
```

The display is left in alpha mode with the cursor at the top of the left margin.

To put the cursor in the home position (top left margin):

```
CALL      HOME
```

Following this call the display is in graphics mode.

To draw a light or dark vector:

```
CALL      PLOT,X,Y
```

where

X = The address of a location containing the X display coordinate

Y = The address of a location containing the Y display coordinate

A dark vector is drawn if the call is immediately preceeded by a call to "GRMODE". Successive calls to "PLOT" draw light vectors between the specified consecutive points.

To output a graphics string:

```
CALL      TTYGRF,CHR
```

where

CHR = Address of the output character string which is terminated with a null.

Following the call, the location "TTYADR" contains the address of the next byte location following the null.

The TTYGRF subroutine is necessary because TTYO and TTYOUT interface to the Tektronix via DOS. The DOS teletype driver does not allow control characters to be printed. Since these characters are necessary for graphics plots, TTYO and TTYOUT cannot be used.

2.3. FILE SYSTEM SERVICES

Extensive services are provided to the programmer for manipulating files. These services include create, retrieve, delete, and rename functions. Accompanying the create and retrieve operations is the ability to read and/or write a file. Standard DOS link and file name blocks are used for the delete and rename functions. A more specialized parameter list called a "request block" has been developed for the create and retrieve functions.

2.3.1. Request Block Format

The request block is generated via the macro "F.REQ" and takes on two general forms - one for the create function and one for the retrieve function. For the create function the macro is invoked as follows:

```
F.REQ     0,LBL,CHR,EXT,RECS,LNG,ACES,FTYP,RTYP,NAME
```

This expands to the following (underlining denotes a calling parameter):

```
LBLCHR:     .WORD       CHR       (address of character string)
            .RAD50      /NAME/    (if parameter "NAME" is non-blank)
LBLNAM:       or
            .WORD       0,0       (if parameter "NAME" is blank)
LBLEXT:     .RAD50      /EXT/
LBLDEV:     .WORD       0         (device name in RAD50 format)
LBLUNT:     .BYTE       0         (device unit number)
LBLRTP:     .BYTE       <4*ACES> + <2*FTYP> + RTYP
                                  (record, file, and access indicator)
LBLBUF:     .WORD       FRECOR    (buffer for file operations)
LBLSIZ:     .WORD       CSIZE2    (size of buffer)
LBLCNT:     .WORD       RECS      (number of records)
LBLLEN:     .WORD       LNG       (record length)
LBLCON:     .WORD       1         (number of consecutive records
                                   required in core simultaneously)
LBLREC:     .WORD       0         (record number requested)
```

The macro call for generating the request block for the retrieve function is:

```
F.REQ     1,LBL,CHR,EXT,ACES,NAME
```

This expands in the same manner as for the create call with the exception that the parameters RECS, LNG, FTYP, and RTYP, which are not specified, are automatically set to zero. The definition of each of the parameters is as follows:

LBL = One to three characters to be used as the first part of the parameter labels.

CHR = Address of the character string to be output to request the file specification. A zero value implies that there is no character string and that the file specification is completely provided in the request block. Therefore, no user interaction will result.

EXT = File extension which indicates the file type with respect to the data contained therein. If this argument is left blank or if the program clears the location associated with this argument, then the filing system uses the extension specified by the user. If the extension is specified via the macro argument or the program, then the user-entered extension is ignored.

RECS = The total number of records to be in the file.

LNG = The length of each record in bytes (odd values are acceptable)

ACES = File access type:

1 = Read
2 = Write
3 = Modify (read and write)

FTYP = File type with respect to structure:

0 = Contiguous
1 = Linked (currently not implemented)

RTYP = Record type:

0 = Fixed length
1 = Variable length (currently not implemented)

NAME = File name. If the name is to be entered by the user at execution time or if the name is to be inserted by the program, this parameter should be left blank. If the name is known at programming time, it should be inserted. If temporary files are to be created, the names FILE1.TMP, FILE2.TMP, ..., FILEN.TMP should be used. These are automatically deleted upon closing the file except when

bit 4 of the LBLRTP parameter is set. In this case the file is closed and not deleted. Caution should be exercised when using this option since .TMP files must all be deleted when the overlay exits.

All request block parameters should be referenced by their respective labels. Future system changes or additions may result in a different order or number of parameters. Therefore, if in a given instance one parameter must be referenced relative to another, then the values defined by the macro "EQUATE" should be used. These values are simply indexes into the request block relative to the first parameter in the block. They are referenced via the last three characters of the label of their respective parameters. For example, if the buffer length parameter must be referenced with respect to the buffer address parameter, then the programmer would use

LBLBUF+SIZ-BUF

as the relative address.

2.3.2. Creating Files

Two file system subroutines are available for creating files. The only difference between them is that one creates the file and then opens it for access while the other simply creates the file. The request block parameters must be initialized prior to either call as discussed above. The one parameter which is of no concern at this point is the record number requested (LBLREC). This is used by other file functions. Special note should be taken of the file system feature which requests the file name from the user. This option is enabled or disabled as explained for the character string address (LBLCHR).

To create a file and open it for access:

```
F.CRE     ADR,ALTRET
          (Ref. Programs EXEC and FILE1)
```

where

ADR = Address of the applicable request block.

ALTRET = Address of an alternate return to be taken if the user responds with a carriage return to the request for a file name. This parameter is optional and need not be specified. When it is not specified a carriage return alone is reported as a recoverable error.

To create a file (file is not opened):

```
F.CRE$    ADR,ALTRET
          (Ref. Programs EXEC and FILE1)
```

where the parameters are as defined for the F.CRE macro.

Both of the above macros generate the appropriate subroutine call to the file system. After each call R0 contains the memory address of the file header. If any data is to be inserted into the header, it should be done at this point (the file access must be set for write or read/write). The first file access (see 2.3.4 below) causes the header to be written into the disk file and to be removed from memory. For the F.CRE$ call, the header is provided for reference only (the header is written into the file prior to return).

It should be noted that the buffer specified in the request block following an F.CRE call is initialized for data I/O and should not be modified. This buffer is not used for the F.CRE$ call. Instead, the header is loaded into a buffer in the executive. This buffer is overwritten by the next F.CRE$, F.RET$, OR F.CHK call.

2.3.3. Retrieving Files

Files are retrieved in a manner similar to that explained for the create option above. Again two options are available, differing only by the "open" operation. The request block is prepared as instructed above. In this case, the number of records, the record length and the file and record types are return parameters. The instructions for the other parameters are the same. The option is also available to have the file system request the file name from the user.

To retrieve a file and open it for access:

```
F.RET     ADR,ALTRET
          (Ref. Programs EXEC and FILE1)
```

where the parameters are identical to those in the F.CRE call above.

To retrieve a file (file is not opened):

```
F.RET$    ADR,ALTRET
          (Ref. Programs EXEC and FILE1)
```

where the parameters are identical to those in the F.CRE call above.

Operations with reference to the file header and the specified buffer are as explained for the create operations. In this case, the header is provided only for reference following the F.RET$ call. The buffer is used by F.RET but not by F.RET$.

2.3.4. Accessing File Data

Following a create or a retrieve function that also opens a file, the file may be accessed via the file system. Prior to each call for a record or group of records, the record requested parameter "LBLREC" should be set to the desired record. The first record in a file is record 1. The calling sequence to request a record or group of records is:

```
F.PTR     ADR,END
          (Ref. Programs EXEC and FILE2)
```

where

ADR = Address of applicable request block.

END = Address of a location to which control is to be returned if one or more of the records requested is non-existent. If set to zero, the filing system will report a fatal error (E1.10.) when a non-existent record is encountered.

Following each call to F.PTR, memory addresses of the first byte of each record requested are found on the stack. For example, if the programmer requests one record (parameter LBLCON), the address of the first byte of that record is found on top of the stack. If three records are requested, then three addresses are returned. In this case the address of the first of the three records is found on top followed, in order, by the other two. The number of consecutive records can be set to any number but must not exceed the value specified at the time that the file was opened.

2.3.5. Check for File Existence

Certain operations simply require that a file's existence be verified or that its parameters be retrieved. The following routine performs these functions. The request block is initialized as for file retrieval. The call is:

```
F.CHK     ADR,NOFILE
          (Ref. Programs EXEC and FILE1)
```

where

ADR = Address of the applicable request block.

NOFILE = Address of an alternate return if the file does not exist.

If the file exists, its number of records and record length are returned in the request block. Also, R0 contains the memory address of the file header. As with F.CRE$ and F.RET$, the executive buffer is used instead of the buffer specified in the request block. Therefore, the desired header information should be extracted prior to any similar operation.

2.3.6. Delete and Rename Functions

As previously mentioned, the delete and rename functions use the standard link and file name blocks as defined by DOS. The format of the call for the delete function is:

```
F.DEL    ADR,NOFILE
         (Ref. Programs EXEC and FILE3)
```

where

ADR = The address of a twelve-word buffer. The first five words of this buffer contain the link block and the remaining seven words contain a file name block. Together they specify the file to be deleted and the device on which it is located.

NOFILE = The address of a location to which the filing system is to return control if the file is non-existent. If this address is zero, then, upon encountering a non-existent file, the execution of the program is terminated, an error message is printed, and control is returned to the executive.

The format of the call for the rename function is:

```
F.RNM    ADR,NOFILE,DUPFIL
         (Ref. Programs EXEC and FILE3)
```

where

ADR = The address of a nineteen-word buffer. This buffer must contain in order a five-word link block, a seven-word file name block for the existing name, and a seven-word file name block for the new name.

NOFILE = As defined for F.DEL above.

DUPFIL = Same as NOFILE except that it applies in the case when a duplicate file name is encountered.

2.3.7. Closing Files

All files that have been opened must be closed following processing. Two calls are provided for this function. First, a single file may be closed via the following call:

```
F.CLOS   ADR
         (Ref. Programs EXEC and FILE3)
```

where

ADR = Address of the applicable request block.

A second available call will close all files that are currently open:

F.SHUT

(Ref. Programs EXEC and FILE3)

If control is returned to the executive due to a fatal error occurring during program execution, any open files will be automatically closed. Any such files that were open for write or modify access will be marked in the fifth word of the header as having been prematurely closed. Subsequent accesses to these files will cause the filing system to warn the user of the file state and allow him to select an alternate file.

2.3.8. Extend a File

A function is provided to allow the length of a contiguous file to be extended. This function should only be used when necessary because it must create a new file, copy the old file to the new and delete the old file. This can be a time-consuming process if the file is large. When required, it is called as follows:

F.EXT ADR,EXTADR

where

ADR = Address of the request block for the file.

EXTADR = Address of a location containing the size of the extension in 256 word blocks.

Note: The file must be in a closed state.

2.3.9. Direct DOS File Access

Although accessing DOS directly for file services is acceptable, it should be avoided if possible. For most applications, sufficient file services have been provided. At times, however, certain requirements may make it necessary to deal directly with DOS. To this end, several functions have been provided in the image processing system. These functions should be used to accomplish direct DOS file access so that the image processing executive can keep track of all initialized and opened data sets. In this manner, the executive can remove them from memory if the execution of the associated option overlay is terminated prior to its normal completion.

To ".INIT" a data set:

```
CALL      F.INIT,LNKBLK
```

where

LNKBLK = Address of the associated link block.

To ".OPEN" a file:

```
CALL      F.OPEN,LNKBLK,FILBLK
```

where

LNKBLK = Address of the associated link block.

FILBLK = Address of the associated file block. The "how open" code must be inserted in the filename block.

To ".CLOSE" a file:

```
CALL      F.CLSE,LNKBLK
```

where

LNKBLK = Address of the associated link block.

To ".RLSE" a data set:

```
CALL      F.RLSE,LNKBLK
```

where

LNKBLK = Address of the associated link block.

2.4. CORE RESIDENT BUFFERS AND PARAMETERS

Certain buffers and parameters are made available within the executive, thus avoiding the need to define them in each overlay. The main working buffer for file I/O begins at the label "FRECOR". The size of this buffer in words is defined by the symbol "CSIZE" and in bytes by the symbol "CSIZE2". The first memory location following this buffer is given by the label "ENDCOR".

As previously discussed, a buffer is available within the executive for keyboard input. This buffer is 80 bytes long and begins at "TTYBUF". The word immediately preceding this buffer is permanently set to the size of the buffer and should not be changed.

Two character strings for requesting image file names are core resident. These strings are made available to promote consistency and eliminate redundancy. They are:

```
INNAM:    .ASCIZ    /INPUT IMAGE NAME=/
```

and

```
OUTNAM:   .ASCIZ    /OUTPUT IMAGE NAME=/
```

Since the frequency of access of other file types is much lower, character strings have not been provided.

2.5. ERROR REPORTING

A centralized error reporting scheme has been adopted in the system. Errors are reported by the programmer via a subroutine call accompanied by the desired error class and error number. Two classes of errors are provided - fatal and recoverable. Fatal errors are those which make it impossible for the program to continue. When such an error call is made, the error reporting routine prints the specified message and passes control to the executive. Therefore, the programmer should not provide any return code.

Recoverable errors are those from which a recovery can be made, such as the user retyping an input parameter. In this case, control is returned to the calling program after the error is printed.

Within each error class exists a list of error messages. Each message is referenced by its corresponding number. The programmer simply passes the number of the error to the error reporting routine. The list of errors can be found in Appendix D.

2.5.1. Fatal Error Reporting

```
ER.FAT    NUM
          (Ref. Program EXEC)
```

where

NUM = The error number within the fatal error class.

2.5.2. Recoverable Error Reporting

```
ER.REC    NUM
          (Ref. Program EXEC)
```

where

NUM = The error number within the recoverable error class.

2.6. UTILITY ROUTINES

A variety of utility routines are available to support the programmer. The function of and access to each routine is discussed below:

2.6.1. Converting Double Word Binary to Decimal ASCII

A routine is available to convert double word binary numbers to decimal ASCII character strings. Two modes of access are available. One mode passes all parameters in general registers and the other mode passes them in a parameter list following the subroutine call. The ASCII string is returned within twelve bytes, right justified, with leading zeroes blank. The minus sign, if present, is right justified with the number. After conversion the number of non-blank characters returned is available at the location "DIGCNT". The register mode call is as follows:

```
CALL    DI2DAR
        (Ref. Program CB2DA)
```

The following registers must be loaded prior to the call:

R2 = Address of the buffer in which to return the ASCII string.

R3 = High order part of double word value.

R4 = Low order part of double word value.

Parameter list mode:

```
CALL    DI2DA,DBLINT,BUFADR
        (Ref. Program CB2DA)
```

where

DBLINT = Address of double word integer (low order part first).

BUFADR = Address of a twelve-byte buffer in which to return the ASCII string.

2.6.2. Converting Single Word Binary to Decimal ASCII

Single word binary conversion is also available. As with double word conversion, two modes are available for passing parameters. One is via general registers and the other is via a parameter list following the subroutine call. The ASCII string is returned within seven bytes, right justified, with leading zeroes blank. The minus sign, if present is right justified with the number. After conversion, the number of non-blank characters returned is available at the location "DIGCNT". The register mode call is as follows:

```
CALL    SI2DAR
        (Ref. Program CB2DA)
```

The following registers must be loaded prior to the call:

R2 = Address of the buffer in which to return the ASCII string.

R4 = Binary value to be converted.

Parameter list mode:

```
CALL    SI2DA,SNGINT,BUFADR
        (Ref. Program CB2DA)
```

where

SNGINT = Address of single word integer.

BUFADR = Address of a seven-byte buffer in which to return the ASCII string.

2.6.3. Converting Floating Point to ASCII

Two conversion methods are available which correspond in function to the FORTRAN "F" and "E" formats. The "F" format returns a number with the decimal point fixed in its proper position. The "E" format returns a number with a decimal point plus a power of ten. The calling sequences follow (due to the .BYTE parameters, the CALL macro may be inconvenient):

F format conversion:

```
JSR     RS,FPASCF           (Ref. Program FPASC)
BR      .+10.
.BYTE   W
.BYTE   D
.WORD   ARG
.WORD   (Continuation of ARG)
.WORD   BUF
```

where

W = The width of the field of characters to be returned.

D = The number of digits desired to the right of the decimal point.

ARG = The two-word floating point number.

BUF = Address of the buffer in which the characters are to be returned.

E format conversion:

```
JSR     RS,FPASCE          (Ref. Program FPASC)
BR      .+10.
```

(The parameters and their descriptions are identical to those for FPASCF.)

If the format is unacceptable, all characters in the return buffer are set to *'s. In "E" format, two *'s are returned for the exponent if it is greater than 99. Either error condition causes the "C" bit in the processor status register to be set. This bit is otherwise cleared.

2.6.4. Convert Internal ASCII Strings to Binary

Two routines are available for converting numeric ASCII strings to binary. They are called as follows:

To convert decimal ASCII to binary:

```
CALL    CONV10
```

To convert octal ASCII to binary:

```
CALL    CONV8
```

For both routines, the address of the ASCII string must be provided in R2. The binary result is returned in R1. The ASCII string can be terminated by a null or a comma.

2.6.5. Convert Radix 50 Packed Characters to ASCII

This routine converts radix 50 packed characters to ASCII. The call is as follows:

```
CALL    RAD2AS,RAD50,ASCII,COUNT
        (Ref. Program RAD2AS)
```

where

RAD50 = Address of the first word in the radix 50 packed string of words.

ASCII = Address of a buffer in which the ASCII string will be returned.

COUNT = The number of radix 50 packed words.

2.6.6. Save and Restore General Registers

This routine saves and restores general registers R0 through R5 using the stack. No parameters are passed.

Saving registers:

```
CALL    SAVREG
        (Ref. Program SAVER)
```

Restoring registers:

```
CALL    RSTREG
        (Ref. Program SAVER)
```

2.6.7. Save and Restore Floating Point Registers

This routine saves and restores the floating point registers AC0 through AC5, the floating point processor interrupt vector, and the floating point status register using the stack. No parameters are passed.

Save FPU status:

```
CALL    SAVFPS
        (Ref. Program SARFPS)
```

Restore FPU status:

```
CALL    RSTFPS
        (Ref. Program SARFPS)
```

2.6.8. Square Root of a Double Word Integer

This routine calcualtes the square root of a double word integer. All registers are unaffected by this routine. The call is:

```
CALL    DPSQRT,DBLINT
        (Ref. Program SQRT)
```

where

DBLINT = Address of the double word integer (low order first). Upon return the square root is found in the first word of the double word. The second word is zero.

2.6.9. Square Root of a Single Word Integer

This routine calcualtes the square root of a single word integer. All registers are unaffected by this routine. The call is:

```
CALL    SQRT,SNGINT
        (Ref. Program SQRT)
```

where

SNGINT = Address of the single word integer. Upon return this location contains the square root.

2.6.10. Partitioning Core

This routine partitions the FRECOR buffer into specified fractional parts and loads the address and size of each partition into their corresponding file request blocks. The call is (a macro is not provided):

```
JSR     R5,PARCOR
BR      .+(n+1)*4
.WORD   DENOM
.WORD   NUMER1
.WORD   BUFAD1
.WORD   NUMER2
.WORD   BUFAD2
   .
   .
   .
.WORD   NUMERn
.WORD   BUFADn
```

where

DENOM = The denominator of the fractional partitions. Each partition is expressed as a fraction of the total; therefore, the denominator is the same for all partitions.

NUMER1-NUMERn = The numerators for expressing the size of the n partitions.

BUFAD1-BUFADn = The n addresses of the buffer address parameters in the respective request blocks.

The CALL macro was not shown here due to the fact that it is limited to six parameters. However, for partitioning calls of six parameters or less, it can be used.

2.7. LOADING OPTION OVERLAYS

The executive provides the capability for one option overlay to request that another be loaded and executed. This should not be confused with the autoload overlay facility. In this case, the called overlay completely destroys the calling overlay. The only return that can be made to the first overlay is for the second one to initiate a request to the executive to load the first one. The second overlay is then destroyed when the first one is loaded. The overlay call request is accomplished via the following macro:

```
LOAD    NUM,ENT
```

where

NUM = Three ASCII characters corresponding to the last three characters (overlay number) of the file name of the overlay to be loaded.

ENT = The number of the entry point (0 is the first entry point) to be used in the called overlay. Note that this parameter allows any addressing mode, i.e., the first entry point may be specified as "#0".

2.8. MISC. MACROS

This section discusses a number of macros which may be useful to the programmer. Use of these macros is at the option of the programmer with the exception of the EQUATE macro. This macro should always be used when the services it provides are required. Refer to DOS documentation for detailed information concerning DOS parameter blocks.

2.8.1. Partitioning Core Buffers

This macro partitions the specified core area into two equal-length buffers and deposits the start address and size of each in their respective request blocks. The user specifies the core space which is to be partitioned. The "FRECOR" buffer is used as a default value.

```
CORD2   ADDR1,ADDR2,NOTLOW
```

which expands to

```
EQUATE                          (macro as defined below)
MOV     #FRECOR,R4              (included only if NOTLOW is blank)
MOV     #CSIZE,R3
MOV     R4,ADDR1
MOV     R3,ADDR1+SIZ-BUF
ADD     R3,R4
MOV     R4,ADDR2
MOV     R3,ADDR2+SIZ-BUF
```

where

ADDR1 = The address of the buffer address parameter within the first request block.

ADDR2 = The address of the buffer address parameter within the second request block.

NOTLOW = A dummy value. If specified, the programmer must provide the core area to be partitioned and its size in words in registers R4 and R3, respectively. If this parameter is left blank, the values "FRECOR" and "CSIZE" are loaded into R4 and R3, respectively.

SIZ and BUF= As defined for the EQUATE macro below.

2.8.2. Move a Specified Number of Bytes

This macro generates code to move a specified number of bytes from one location to another.

```
BYTMOV  FROM,REG1,TO,REG2,CNT,REG3
```

which expands to

```
          .ENABL    LSB
          MOV       #FROM,REG1
          MOV       #TO,REG2
          MOV       #CNT,REG3
1$:       MOVB      (REG1)+,(REG2)+
          SOB       REG3,1$
          .DSABL    LSB
```

where

FROM = Address of first byte to be moved.

TO = Address to which the first byte is to be moved.

CNT = The number of bytes to be moved.

REG1,REG2 & REG3 = General registers.

2.8.3. Definition of System Parameters

This macro defines the system parameters via equate statements.

EQUATE (no parameters)

which expands to (all numbers are octal except where a decimal point appears)

```
CRE     =    0
RET     =    1
RD      =    1
WR      =    2
MO      =    3
CONTIG  =    0
LINKD   =    1
FLEN    =    1
VLEN    =    1
CR      =   15
LF      =   12
CHR     =    0
NAM     =    2
EXT     =    6
DEV     =   10
UNT     =   12
RTP     =   13
BUF     =   14
SIZ     =   16
CNT     =   20
LEN     =   22
CON     =   24
```

```
REC     =  26
LINMAX  =  33
HDRSIZ  =  62
```

where the following pertains to specifying a request block:

CRE = Specifies the create function.

RET = Specifies the retrieve function.

RD = Specifies read only access.

WT = Specifies write only access.

MO = Specifies read and write access.

CONTIG = Specifies a contiguous file.

LINKD = Specifies a linked file.

FLEN = Specifies a fixed length record.

VLEN = Specifies a variable length record.

The following define ASCII codes:

CR = ASCII carriage return.

LF = ASCII line feed.

The following define the positions of the parameters within the request block relative to the beginning of the block:

CHR = Character string address parameter.

NAM = File name parameter.

EXT = File extension parameter.

DEV = Device parameter.

UNT = Unit number parameter.

RTP = Record type parameter.

BUF = Buffer address parameter.

SIZ = Buffer size parameter.

CNT = Number of records parameter.

LEN = Record length parameter.

CON = Number of contiguous records parameter.

REC = Record requested parameter.

The following refers to frame displays:

LINMAX = Number of lines of dialogue to allow before rebuilding the display.

The following refers to the header established by the file system within the buffer specified in a request block:

HDRSIZ = Size of file I/O buffer header in bytes.

2.8.4. Definition of Floating Point Registers

This macro defines six mnemonics for use in referencing the floating point processor registers.

```
REGS      (no parameter list)
```

which expands to

```
AC0  =    %0
AC1  =    %1
AC2  =    %2
AC3  =    %3
AC4  =    %4
AC5  =    %5
```

where

AC0 through AC5 = Floating point register mnemonics.

2.8.5. Inserting File Header Text

This macro is used to insert a character string into a file header. The memory address of the beginning of the header is expected to be in R0. Up to three separate character strings can be specified for transfer into the header.

```
HDRTXT    ADR1,ADR2,ADR3
```

which expands to

```
ADD       #384.,R0
MOV       ADR1,R1
MOVB      (R1)+,(R0)+
BNE       .-2
```

```
DEC     R0
MOV     ADR2,R1         } This code is generated
MOVB    (R1)+,(R0)+     } only if ADR2 is not blank
BNE     .-2
DEC     R0
MOV     ADR3,R1         } This code is generated
MOVB    (R1)+,(R0)+     } only if ADR3 is not blank
BNE     .-2
```

Note that any addressing mode is legal for ADR1, ADR2 and ADR3. For example, if the lable "MS0" is attached to a character string its address is specified by "#MS0".

2.8.6. Pushing and Popping Stack Items

These two macros simply push or pop up to six items to or from the stack. If any parameter is omitted, its corresponding line of code is not generated.

To push items onto stack:

```
PUSH    A,B,C,D,E,F
```

which expands to

```
MOV     A,-(SP)
MOV     B,-(SP)
MOV     C,-(SP)
MOV     D,-(SP)
MOV     E,-(SP)
MOV     F,-(SP)
```

To pop items from stack:

```
POP     A,B,C,D,E,F
```

which expands to

```
MOV     (SP)+,A
MOV     (SP)+,B
MOV     (SP)+,C
MOV     (SP)+,D
MOV     (SP)+,E
MOV     (SP)+,F
```

2.8.7. Defining a DOS Link Block

```
LNKBLK  CHR,RET,LDN,NWTF,UN,PDN
```

which expands to (underlining indicates a calling parameter):

```
        .WORD   RET
CHRLNB: .WORD   0
```

```
        .RAD50    / /       (if LDN is left blank)
            or
        .RAD50    /LDN/     (if LDN is non-blank)
        .BYTE     NWTF
        .BYTE     UN
        RAD50     / /       (if PDN is left blank)
            or
        .RAD50    /PDN/     (if PDN is not blank)
```

where

CHR = One to three characters to be used as the first characters of the parameter label.

RET = Error return address if the data set cannot be initialized.

LDN = Logical device name as defined under DOS.

NWTF = Number of words to follow (physical data set name is the first word).

UN = Unit number.

PDN = Physical data set (device) name as defined under DOS.

2.8.8. Defining a DOS File Block

```
        FILBLK    CHR,RET,HO,FILNAM,EXT,UIC,PC
```

which expands to

```
        .WORD     0
        .BYTE     HO
CHRERC: .BYTE     0
        .WORD     0,0,0     (if FILNAM is left blank)
CHRFNB:     or
        .RAD50    /FILNAM/  (if FILNAM is not blank)
        .RAD50    /EXT/
        .WORD     UIC
        .WORD     PC
```

where

CHR = One to three characters to be used as the first characters of the parameter labels.

RET = Error return address if the file operation is unsuccessful.

HO = How open code (see DOS documentation for codes).

FILNAM = Six-character file name.

EXT = Three-character file name extension.

UIC = User ID code.

PC = Protection code.

2.8.9. Defining a DOS Buffer Header

```
BUFHDR    CHR,MBCNT,MODE,ABCNT,PNTR
```

which expands to

```
CHRMBC:   .WORD    MBCNT
          .BYTE    MODE
          .BYTE    0
CHRABC:   .WORD    ABCNT
          .WORD    PNTR      (appears only if PNTP is not blank)
```

where

CHR = One to three characters to be used as the first three characters of the parameter labels.

MBCNT= Maximum byte count in buffer.

MODE = Data transfer mode.

ABCNT = Actual byte count for transfer.

PNTR = Starting address of buffer for dump mode.

2.8.10. Defining a DOS TRAN Block

```
TRNBLK    CHR,DBN,BUFR,WCNT,FUNC
```

which expands to

```
CHRTNB:   .WORD    DBN
CHRBUF:   .WORD    BUFR
CHRCNT:   .WORD    WCNT
          .BYTE    FUNC
CHRTST:   .BYTE    0
          .WORD    0
```

where

CHR = One to three characters to be used as the first three characters of the parameter labels.

BUFR = Address of buffer for transfer.

WCNT = Number of words to transfer.

FUNC = Function to perform.

2.8.11. Defining a DOS Record Block

```
RECBLK    CHR,FUNC,BUF,RLEN,HORN,LORN
```

which expands to

```
CHRRCB:   .BYTE     FUNC
          .BYTE     0
          .WORD     BUF
          .WORD     RLEN
          .WORD     HORN
          .WORD     LORN
```

where

CHR = One to three characters to be used as the first three characters of the parameter labels.

BUF = Address of buffer for transfer.

RLEN = Record length in bytes.

HORN = Record number (high order).

LORN = Record number (low order).

Defining a BLOCK Block

```
BLKBLK    CHR,FUNC,BNUM,BUF,LEN
```

which expands to

```
CHRBKB:   .BYTE     FUNC
          .BYTE     0
CHRBNM:   .WORD     BNUM
          .WORD     BUF
          .WORD     LEN
```

where

CHR = One to three characters to be used as the first three characters of the parameter labels.

FUNC = Function to be performed.

BNUM = Requested block number.

BUF = Address of buffer for transfer.

LEN = Length of buffer.

2.9. LINKING THE USER PROGRAM

The user-developed program is easily linked for execution under the image processing system. Any name can be assigned to the load module. The linking sequence is as follows:

```
name,LP: <IPS.STB,user-supplied programs,LIBR/T:35700/E
```

The library module (LIBR) is required only if routines contained therein are referenced. The top switch (/T:35700) is dependent upon the value used for building the system (Ref. Appendix G). The value given here is that used for implementation in 28K words of memory.

The overlay autoload facility is fully described in "DOS/BATCH Linker (LINK) Programmer's Manual". In this case, all input file names appear in an overlay descriptor file which is also described in the "LINK" manual. The symbol table file "IPS.STB" must be specified in the overlay descriptor file. The autoload software is linked with the executive module and therefore need not be specified.

The user program is then executed through the "CALL USER PROGRAM" option in Frame 13. The program is then referenced by its load module name. The user ID number is required if the program was linked under another UID. The device is assumed to be the system software device.

An entry point must also be entered when executing the program. The first entry point is "1" and is specified in the program via the ".END" assembler directive. The second entry point is the first entry plus one word.

The program must exit to the executive upon completion via an "RTS R5".

2.10. DEBUG FACILITY

The DOS program "ODT" can be used for program development. However, due to the limited space for option overlays a special system must be built in which ODT is made memory resident. This system is built using the batch stream contained in files IPSODT.BAT and IPSBLD.BAT. Appendix G provides details for this procedure.

After this system is built the user program can be linked with the new system symbol table.

Program execution using ODT is then effected in a special way. The following steps must be taken:

1. Use the DOS GET command to load the image processing system:
   $GET IPS

2. Call ODT:
   $OD

3. Set a breakpoint:
   *nnnnnn/B
   where nnnnnn is the address of the label E.OVR found in the IPS.STB section of the user program load map.

4. Start system:
   *mmmmmm/G
   where mmmmmm is the start address given by the label E.EXEC found in the IPS.STB section of the user program load map.

The system is then used in the normal manner. When any option is selected that causes an option overlay to be loaded, the previously set breakpoint is encountered. This breakpoint appears at a point in the executive after the overlay has been loaded, but just prior to its execution. Therefore, for regular system overlays, the ODT proceed command (;P) can be given.

The user program is executed in the normal fashion via the "CALL USER PROGRAM" option. When this option is selected, the breakpoint will be encountered. The proceed command should be issued. The program name and entry point are then entered after which the breakpoint will again be encountered. At this point, the user program is in memory and breakpoints can be set within for debugging.

SECTION 3

PERMANENT SOFTWARE ADDITIONS

The image processing system has been designed as a completely open-ended system with respect to new software additions. The procedures to be followed, which are quite simple and straightforward, are discussed in the following paragraphs.

3.1. BUILDING THE NEW OPTION OVERLAY

The suggested procedure for developing new software prior to its permanent incorporation is to follow the guidelines for creating a user program given in Section 2. When it is operating correctly and has been determined to be of general interest to system users, it can then be permanently incorporated into the operating system.

The first step is to link the program with needed subroutines to form the overlay load module. The linking sequence is identical to that for the user program except that a special name must be used and the output load module must be stored under the user ID number that is used for the system software. The name must consist of six characters where the first three are "OVR." The last three characters are formed from the number of the overlay. This can be any number that is not currently used (ref. Appendix G). Typically, these numbers are assigned sequentially for ease of system management.

Consider, for example, that an overlay is being linked from a routine named "NEWJOB" which references library subroutines. Let us assume that the next available overlay number is 83. The link sequence is then entered as **follows:**

```
OVRØ83/CO,LP: < IPS.STB, NEWJOB, LIBR/T:357ØØ/E
```

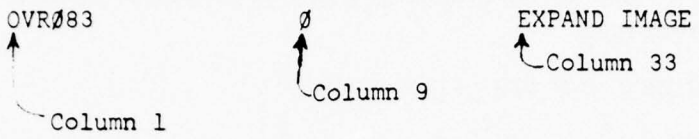
The switch "/CO" is used to force the output file to be created in contiguous format. By doing so, slightly faster overlay load times are realized. The top switch is set as discussed in Section 2.

This completes the overlay building task. The programmer can now proceed to the next task which is entering the overlay option into a frame.

3.2. ENTERING A NEW OPTION INTO A FRAME

The second and last step in permanently adding a new system option is to create a new entry in the appropriate frame. The first task then is to select the frame to which the option will be added. Each frame contains options which are closely related in function. The new option should therefore be added to the frame which describes its function.

The entry is made by editing the file "FRAMES.SRC" or by modifying the frame card deck and then loading this deck into "FRAMES.SRC" via PIP (Ref. Appendix C). In either case the option entry is made in the same format. For purposes of illustration let us assume that an entry is being created for the previously linked overlay "OVRØ83" and that the option is to appear as "EXPAND IMAGE" in the frame display. The entry is made as follows:



The column numbers refer to card columns. When using the editor to edit the file, spacing should be accomplished with blanks. Tab characters are not accepted by the frame building software.

The first entry beginning in column one is the name of the file in which the overlay is stored. The entry in column nine is the entry point number. The first entry point is specified by "Ø", the second "1", etc. The last entry begins in column 33, which is the text that appears in the frame display. Note that the option number is not specified. The frame display software assigns the option numbers on the basis of order of appearance.

After the file FRAMES.SRC has been prepared with the new entry, the frame formatting software must be called. This is accomplished under DOS by entering the following command:

```
RUN FRAMES
```

When the formatting is complete the following is printed:

```
FRAMES.IPS CREATED
```

The newly modified system is then ready for use.

The function of the "FRAME" program is to convert the "FRAMES.SRC" file which is in the user-oriented format to the "FRAMES.IPS" file which is in the system format (Ref. Appendix C). This procedure greatly simplifies frame updates.

3.3. CREATING A NEW FRAME

It may be necessary to create another system frame if new software does not fall into any of the existing frame categories. In this case several new lines must be entered into the FRAMES.SRC file. The different types of line entries will be discussed in the following paragraphs. The programmer is referred to Appendix C for the current FRAMES.SRC file listing which can serve as a guide to creating the new frame.

The first line type is simply a comment. This is indicated by a **semicolon** in column one. A blank in column one has the same effect. In either case the line content is ignored by the FRAMES program.

Another line that has no effect on the final product consists of only a form feed character in column one. To allow such a line to be specified in the card deck, a special control card heads the deck. This control card causes the card reader input to be interpreted as ASCII format.

The third line type specifies the total number of frames in the file. This line contains only one entry which is a number beginning in column one. Only one line of this type can occur in the file and it must precede all frames. If a new frame is to be added, this number must be modified.

The next line type, is the frame header. The first three columns must contain the characters "FRM". The next three characters must consist of the frame number. The last entry on this line type is the frame header text which begins in column 33.

This brings us to the frame option lines which may be one of two types. The first type causes a transfer to a new frame. It is indicated by the characters "TRNFRM" in columns one through six. The number of the frame to be displayed by the option begins in column 9 and the text to be displayed for the option begins in column 33.

The other option type causes an overlay to be loaded and executed. In this case the first six columns contain the file name of the overlay. The overlay entry point number which corresponds to the option begins in column nine. As in the previous case, the option text begins in column 33.

One final line type remains to be discussed. This type signals the end of the frame and consists of simply the characters "END" in the first three columns.

From these various line types the new frame must be constructed. The next consecutive frame number that is not in use should be assigned to the frame. The final task is to enter the name of the frame as an option in frame Ø and to execute the FRAMES.SRC program.

3.4. ADDING A NEW ERROR MESSAGE

If a new error message is required, it can be added in a manner similar to adding a frame entry. In this case the file ERRORS.SRC is edited via the editor or by changing its associated card deck. The program "MESAGE" is then executed via the DOS RUN command to form the file ERRORS.IPS. The format of this file and the current contents of the ERRORS.SRC file are given in Appendix D.

Each new error must be assigned a type and a unique number. Recoverable error types are prefixed with "EØ.n" where n is the error number. Fatal errors have the prefix "El.n". The errors must first be ordered sequentially in the file by type and then by number.

SECTION 4

PROGRAM DOCUMENTATION

A description of each program in the system is contained on the following pages. These programs are one of two types. The first type is designed to be called by the executive and therefore the calling sequence is listed simply as one or more entry points. These routines return control by executing an "RTS R5" instruction.

The second program type consists of one or more callable subroutines. The subroutine calling sequences are detailed complete with calling parameters and descriptions sufficient for the programmer to call upon their services.

The documentation in this section is not meant to describe the detailed methods by which the programs accomplish their assigned tasks. It is intended to be a concise reference to all programs contained within the system. The detailed documentation is included within the listings which have been commented extensively.

Program Name: ARSLCT

Purpose:

To create an image file from a specified rectangular area within an existing image file.

Description:

This routine creates an image from a user-specified area within an existing image. Specification of the area is obtained through the keyboard via the subroutine "SNGDEC". All of the available free core buffer area is used for reading the existing image, extracting the specified area and writing the new image onto the disk.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, PARCOR, RAD2AS, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

ARSLCT

Program Name: AS2RAD

Purpose:

To pack an ASCII character string into RAD5Ø.

Description:

An ASCII character string is packed into RAD5Ø. This mode of packing is fully described in the "DOS Monitor Handbook". Three characters are packed at a time via the DOS ".RADPK" programmed request. An error return is taken if any illegal characters are found.

Subroutines Called:

RSTREG and SAVREG

Calling Sequence:

```
JSR       R5,AS2RAD
BR        .+10.
.WORD     Address of ASCII string
.WORD     Address for output RAD5Ø string
.WORD     Number of ASCII characters
.WORD     Address of illegal character error routine.
```

General registers are not altered.

Program Name: BLDISP

Purpose:

To retrieve and display an option frame.

Description:

This routine reads the specified frame into core and displays it on the Tektronix graphics terminal. Since the frame text is packed into RAD5Ø format, it must be converted to ASCII prior to its display. This unpacking is accomplished via the DOS programmed request ".RADUP". The frame itself is read from the disk via the request ".TRAN". Registers are saved upon entry and restored upon exit to prevent interference with other operations.

A second routine is available which simply reads the frame from the disk and unpacks the text.

Subroutines Called:

ALPHA, CLEAR, E.ERR, GRMODE, HOME, PLOT, RSTREG, SAVREG and TTYO

Calling Sequence:

Read, unpack and display frame:

```
JSR  R5,BLDISP
```

Read and unpack frame:

```
JSR  R5,UNPACK
```

The following globals are set by EXEC:

FRMNUM - Set to the number of the desired frame

FRMADR - The address of the first block of the "FRAMES.IPS" file

FRMTBL - Prior to the first call to one of the routines the first word of this buffer must be set to a number other than that of the frame specified in FRMNUM.

General registers are not altered.

Program Name: CALUSP

Purpose:

To load and execute a user program.

Description:

The name of the routine is obtained from the user via "GETNAM". The disk is then searched through the ".LOOK" programmed request to insure the program's existence. The file specification is then loaded into the link and file blocks (RUNLNK and RUNFIL) within EXEC. Next the entry point is obtained and control is transferred to the overlay loading routine in EXEC.

Subroutines Called:

E.ERR, GETNAM, LOADER and SNGDEC

Calling Sequence:

Entry is from the Executive at:

```
CALUSP
```

Program Name: CB2DA

Purpose:

To convert single or double word binary numbers to decimal ASCII.

Description:

Two modes of parameter passing are offered for complete flexibility. They can be passed in general registers or in parameter lists. For efficiency this routine was programmed to use the floating point processor. The MODF instruction is extremely convenient, especially for double precision numbers.

Subroutines Called:

RSTFPS, RSTREG, SAVFPS and RSTREG

Calling Sequence:

Convert double word to ASCII:

```
        JSR     R5,DI2DAR

                R3 must contain high order part of number

                R4 must contain low order part of number

                R2 must contain the address of a twelve-byte buffer
                   for the output string
```

or,

```
        JSR     R5,DI2DA

        BR      .+6

        .WORD   Address of double word number (low order part first)

        .WORD   Address of a twelve-byte return buffer.
```

Convert single word to ASCII

```
        JSR     R5,SI2DAR

                R4 must contain the single word integer

                R2 must contain the address of a twelve-byte buffer
                   for the output string.
```

or,

```
        JSR     R5,SI2DA
        BR      .+6
        .WORD   Address of a single word number
        .WORD   Address of a twelve-byte return buffer.
```

General registers are not altered.

Program Name: CHCUR

Purpose:

To capture the position of the Tektronix cross hair cursor.

Description:

This routine turns on the Tektronix cross hair cursor and waits for the user to select a position. After a position is selected and a character is struck this program accepts the input byte stream. This byte stream is then converted to the coordinate values which are placed in the parameter list. The character struck by the user is also returned.

Subroutines Called:

OUTCAR, RSTREG and SAVREG

Calling Sequence:

```
JSR       R5,CHCUR
BR        .+8.
.WORD     Character struck on keyboard
.WORD     X coordinate value
.WORD     Y coordinate value.
```

General registers are not altered.

Program Name: CKTEXT

Purpose:

To examine file header text and allow it to be changed.

Description:

The file is obtained via F.RET. The header text is then printed on the graphics display. Replacement text is then accepted. If none is entered, the file is closed without change.

Subroutines Called:

F.CLOS, F.RET, TTYIN, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

CKTEXT

Program Name: CLCV

Purpose:

To remove all points from a binary image that are not part of a closed curve.

Description:

First the input file is copied to the output file. All further operations are then performed on the output file. The neighborhood of each pixel of value 255 is examined to determine if it is a part of a closed curve. If only one or no neighbors are 255, then the center pixel is changed to zero. If more than one neighbor is 255, then the surround is checked for an "edge, no edge, edge, no edge" condition. If the condition exists, the point is not changed. If it does not exist, the point is changed to zero.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT and RAD2AS

Calling Sequence:

Entry is from the Executive at:

START

Program Name: CMPRES

Purpose:

To compress and expand matrices.

Description:

This program contains two subroutines. The first removes row and column elements of either a matrix or a vector according to a calling parameter list. The second subroutine inserts zeroes into row and column elements of either a matrix or vector, also according to a calling parameter list.

Subroutines Called:

RSTREG and SAVREG

Calling Sequence:

Expand a matrix or vector:

```
JSR       R5,EXPAND
BR        .+1Ø.
.WORD     Address of address of list of row and column elements
          at which to insert zeroes
.WORD     Address of address of matrix or vector
.WORD     Address of number of rows in matrix
.WORD     Address of number of columns in matrix.
```

Compress a matrix or vector:

```
JSR       R5,CMPRES
BR        .+1Ø.
.WORD     Address of address of list of row and column elements
          to remove
.WORD     Address of address of matrix or vector
.WORD     Address of number of rows in matrix
.WORD     Address of number of columns in matrix.
```

General registers are not altered.

Program Name: COMBIM

Purpose:

To compute the average, a scaled difference or the absolute difference of two images.

Description:

Two input images are combined point by point according to the entry point selected. The average is formed by adding the points and dividing by two. The scaled difference is the difference between the points plus 255, all of which is divided by two. The absolute difference is simply the absolute value of the difference between the points.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, PARCOR and RAD2AS

Calling Sequence:

Entry is from the Executive at:

AVE - Average images

AVE+2 - Scaled difference

AVE+4 - Absolute difference

Program Name: COMPIL

Purpose:

To compile logical and arithmetic statements into executable machine code.

Description:

Two entry points to this routine are available. One accepts a logical expression from the keyboard and compiles it into machine code. This was written for the Boolean logic option. The second entry point allows several complete statements to be entered. Three statement types are recognized. The first is simply an arithmetic assignment statement, the second is like a FORTRAN logical IF statement and the third is a modification of the IF called an IFGO. This is simply a logical IF that when satisfied, the arithmetic assignment is evaluated and all following statements are skipped.

Both entry points return the compiled code in the same format: the compiled code, a list of symbols used in the statements, the location within the code of the values represented by the symbols and the ASCII string consisting of the user entered statements.

Subroutines Called:

AS2RAD, E.ERR, RSTFPS, RSTREG, SAVFPS, SAVREG, TTYIN and TTYOUT

Calling Sequence:

To compile assignment, IF and IFGO statements:

```
JSR     R5,COMPIL
BR      .+6
.WORD   Address of a working buffer
.WORD   Size of buffer in words.
```

To compile a logical expression only:

```
JSR     R5,CMPILB
BR      .+6
.WORD   Address of a working buffer
.WORD   Size of buffer in words.
```

The buffer supplied to the two subroutines should be made as large as possible to prevent overflow. Upon return the buffer contains the following:

Program Name: COMPIL (Continued)

Word	Definition
1	Number of words used in buffer.
2	Address of compiled code relative to word 1.
3	Address of symbol table relative to word 1.
4	Address for symbol values relative to word 1.
5+	Input ASCII character string terminated with a null.
	.
	.
	.
	Symbol Table
	.
	.
	.
	Compiled Code
	.
	.
	.
	Symbol Values

SYMBOL TABLE FORMAT

Word	Definition
1	Number of symbols
2	Status word for symbol 1
3, 4	RAD5Ø pack of symbol 1
5	Status word for symbol 2
6, 7	RAD5Ø pack of symbol 2
	.
	.
	.
	Status word for symbol N
	RAD5Ø pack of symbol N

Program Name: COMPIL (Continued)

STATUS WORD FORMAT

Bit Ø Ø = Symbol is not used as an independent variable

1 = Symbol is used as an independent variable

Bit 1 Ø = Symbol is not used as a dependent variable

1 = Symbol is used as a dependent variable

Bit 2 Ø = Non-neighborhood symbol

1 = Neighborhood symbol.

The compiled code is called as follows:

```
JSR     PC,(adr. of code)
```

Prior to this call, the values of all variables used only as dependent variables must be defined. These values are placed in the buffer area identified by word 4 of the return buffer. The first four words of this area are reserved for special subroutine addresses. The first three should be filled with the entry point addresses of the EXP, LN and LOG routines respectively. The fourth is a special routine that retrieves image points whose positions are relative to the current point. This routine must be provided if the code that is entered references it. The routine is called as follows:

```
JSR     R5,(adr. of neigh. ref. routine)
.WORD   Image number
.WORD   Row offset from current position
.WORD   Column offset from current position.
```

(NOTE: This capability is not currently used by the system. It is planned for use in future work.)

Following the four special addresses appear the symbol values in the order of appearance within the symbol table. All values are in two word floating point. Independent variable values will be set upon return.

When "CMPILB" is called the logical result is returned in R1. The returned value is either a 1 for true or a Ø for false.

Program Name: CREFLT

Purpose:

To generate a filter file.

Description:

This routine allows a filter to be generated as a function of the row and column position within the filter array or as a function of the Euclidean distance from the upper left corner of the array. The function is defined by the user at the keyboard. The routine "COMPIL" is called to accept this input and to compile the function into machine code.

Prior to generating the filter file, user specified cross sections of the specified filter function are displayed on the graphics display. These cross sections must be perpendicular to the X-Y plane.

In addition to allowing a filter file to be generated, the specified function can be stored in a separate file. This allows it to be recalled in the future for further filter generation.

Subroutines Called:

BLDISP, COMPIL, ENABLR, EXP, E.ERR, FLTENT, FPSQRT, F.CLOS, F.CRE, F.PTR, F.RET, GETCON, LN, LOG, RAD2AS, SI2DA, SNGDEC, TTYIN, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

CREFLT

Program Name: CRESFS

Purpose:

To create and edit spectral and vector set files.

Description:

This routine performs one of two functions based upon the entry point selected. One entry point deals with spectral sets and the other with vector sets. Similar operations are performed by both routines. File names are entered at the keyboard which are then stored in a file. For spectral file sets, class symbols and data reduction factors are also accepted and stored.

A second operating mode can be entered in which the file name lists can be edited and/or displayed.

Subroutines Called:

BLDISP, CLEAR, DSABLR, ENABLR, E.ERR, F.CLOS, F.CRE, F.DEL, F.PTR, F.RET, F.SHUT, GETNAM, LNPRNT, LPCLOS, RAD2AS, RSTREG, SAVREG, SI2DA, SI2DAR, SNGDEC, TTYIN, TTYO and TTYOUT.

Calling Sequence:

Entry is from the Executive at:

CRESFS - spectral set file operations

CRESFS+2 - vector set file operations

Program Name: CREVEC

Purpose:

To create vector files and to add measurements to existing vector files.

Description:

This routine creates two types of vector files. A design vector file is created from a spectral set in which regions have been described. In this case each vector can be identified as belonging to a particular class. A test vector file is created from all points within the spectral set. Since the vectors are not created on the basis of regions, an identifying class symbol cannot be assigned.

Only spatial measurements can be added to an existing vector file. These measurements consist of basic statistics computed over neighborhoods of each point.

Subroutines Called:

BEGFND, CFREQ, CMEAN, CSDEV, DI2DAR, E.ERR, F.CLOS, F.CRE, F.DEL, F.PTR, F.RET, F.RNM, F.SHUT, FINDPT, GHIGH, GLOW, GMEAN, GMEDN, GRANGE, GSDEV, RAD2AS, SI2DA, SMINIT, SNGDEC, SNGLPT, TTYIN, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

CREVEC - Create test vector file

CREVEC+2 - Create design vector file.

Program Name: CRLOG

Purpose:

To create and initialize logic tree files.

Description:

This routine creates a logic tree file. The directory, the logic tree and the class symbol pages are all initialized according to the current vector set. The overall result is a logic tree consisting of only the senior node. All class symbols are assigned to that node.

Subroutines Called:

E.ERR, F.CRE, F.CLOS, F.PTR, RSTREG and SAVREG

Calling Sequence:

```
        JSR       R5,CRLOG

        BR        .+8.

        .WORD     Address of the address of a working buffer

        .WORD     Address of the size of the buffer

        .WORD     Address of a four word file specification
                  (Device (RAD5Ø), unit and file name (RAD5Ø))
```

Program Name: CTFILE

Purpose:

To create an input file from values entered via the keyboard.

Description:

The values for each pixel within the image are obtained through the keyboard. The values are then stored in a disk file.

Subroutines Called:

F.CRE, F.PTR, F.SHUT, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

CTFILE

Program Name: DECIM

Purpose:

To reduce an image in size by discarding pixels.

Description:

The image is reduced by retaining every Nth row and every Mth column. The result is stored in a disk file.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, RAD2AS, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

DECIM

Program Name: DECIMG

Purpose:

To create a thematic map from a classified set of vectors.

Description:

A thematic map is created to present the classification of vectors in an image format. The first step in the process is to insure that each terminal node in the tree is associated with only one class. Also, the vector file must contain positional information with respect to rows and columns. The classification of each vector is determined and a grey level is assigned to the pixel value associated with the position of the vector. Grey level assignments are made by the user on the basis of class.

Subroutines Called:

CLLF, DI2DAR, E.ERR, F.CRE$, F.PTR, F.RET, F.RET$, F.SHUT, GETNAM, GETNOD, GETSYM, LNPRNT, LPCLOS, OPLF, RAD2AS, SI2DAR and SNGDEC

Calling Sequence:

Entry is from the Executive at:

DECIMG

Program Name: DELETE

Purpose:

To delete a file from a disk peripheral.

Description:

The file specification is obtained from the user. The file is then deleted by calling F.DEL. A loop is contained in the program to allow several names to be entered. For successive names the previous file name extension is used if none is entered.

Subroutines Called:

E.ERR, F.DEL and GETNAM

Calling Sequence:

Entry is from the Executive at:

DELETE

Program Name: EXEC (Continued)

Program Name: DENHST

Purpose:

To compute and display histograms of images and regions within images.

Description:

The histograms computed are based upon the density values within the image and the maximum, minimum and average difference between each point and its eight neighbors. These are computed for complete images and for specified regions within images. The routine "FINDPT" is used to determine which points are within the regions. After the histogram is computed, the routine "HISPLT" is called to display it on the graphics display.

After the histogram is displayed several options are available. Most of these deal with such things as zooms, change tick mark spacing, etc. One option determines the percentage of the total number of points that lie within a user specified range. Another allows a cumulative histogram to be displayed.

In addition to above options, a density profile of any given row can be displayed in histogram format. Display options similar to those discussed above are also available for this function.

Subroutines Called:

ALPHA, BEGFND, BLDISP, CLEAR, DBLDEC, DI2DA, DI2DAR, DSABLR, ENABLR, E.ERR, FPASCF, F.CLOS, F.PTR, F.RET, FINDPT, GRMODE, HISPLT, HOME, OPTION, PLOT, RAD2AS, RSTREG, SAVREG, SI2DA, SI2DAR, SNGDEC, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

BEGIN - Calculate image histograms

BEGIN+2 - Calculate region histograms

BEGIN+4 - Display single line density profile.

Program Name: DICOMD

Purpose:

To record images on film via the Dicomed Image Recorder.

Description:

A user specified image is retrieved and printed on the Dicomed Image Recorder. The recorder resolution, polarity and transfer function are set by this routine according to the user's specification. A blow-up factor is also accepted. The blow-up is accomplished by replicating both rows and columns prior to transfer to the recorder.

Subroutines Called:

E.ERR, FRB, FRI, FRW, F.CLOS, F.CRE, F.PTR, F.RET, F.SHUT, RSTREG, SAVREG, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

DICOMD

Program Name: DIFFAC

Purpose:

To create a scaled, weighted combination of two images.

Description:

Each point in the two input images is multiplied by a factor. The combination is then formed according to the signs of the weights. The result is then scaled over the range of zero to 255.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, PARCOR, RAD2AS, SI2DA, SI2DAR, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

DIFFAC

Program Name: DIRCTY

Purpose:

To print directories of disks on the line printer or graphics display.

Description:

The directory is read into core from the specified disk and is sorted first by extension and then alphabetically. If a short directory is requested, it is then printed on the specified device. If a long directory is desired, the header block of each file is read and then the file size and header text are extracted for inclusion in the listing.

Subroutines Called:

BLDISP, CLEAR, DSABLR, ENABLR, E.ERR, F.INIT, F.RLSE, GETDEV, GETNAM, LNPRNT, LPCLOS, RAD2AS, SI2DA, SNGDEC, TTYIN, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

DIRCTY - List long directory

DIRCTY+2 - List short directory

Program Name: ELCHNG

Purpose:

To replace specified density values or ranges of values with new values.

Description:

The type of operation, individual values or ranges, is determined by the entry point. In either case, the values and ranges along with the replacement values are specified by the user. A table of 256 entries is then created which defines a transfer function over the grey value range of Ø to 255. This table, along with the input and output files, is passed to TRNFTN for processing.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, RAD2AS, SNGDEC, TRNFTN and TTYOUT

Calling Sequence:

Entry is from the Executive at:

ELCHNG - Change individual elements

ELCHNG+2 - Change ranges of elements.

Program Name: EVALBL

Purpose:

To apply Boolean logic to a vector set.

Description:

This routine is called by EVALDR to apply Boolean logic which resides at a given node in the logic tree. The logic file is opened, the logic is extracted and the file is closed. The vector set is then opened and the logic is applied to each vector. The vector set is then closed upon completion.

Subroutine Calls:

CLLF, CLOVEC, EXP, GETVEC, GTLF, LN, LOG, OPLF, OPNVEC, RSTFPS, RSTREG, SAVFPS, SAVREG

Calling Sequence:

```
JSR       R5,EVALBL
```

Prior to entry, the first nine words of the "FRECOR" buffer must be set to:

Word	Description
0	Logic node number
1	Logic file device (RAD50)
2	Logic file unit number
3,4	Logic file name (RAD50)
5	Vector file device (RAD50)
6	Vector file unit number
7,8	Vector file name (RAD50)
9	Node number at which logic evaluation began (used only by EVALDR)

These values are not modified.

Program Name: EVALDR

Purpose:

To control the overall logic evaluation operations.

Description:

This routine accesses the logic file to determine which logic evaluation routine should be called at each node in the tree. That routine is then called.

Three modes of operation exist. The first mode evaluates the logic at the current node only. The second mode evaluates logic at the current node and all nodes below the current node. The last mode evaluates all logic in the tree beginning at the senior node.

Subroutine Calls:

CLLF, CLLOGF, EVALBL, EVALFP, E.ERR, GETNAM, GETNOD, GTLOGF, OPLF and OPLOGF

Calling Sequence:

Entry is from the Executive at:

EVALDR

Program Name: EVALFP

Purpose:

To apply Fisher pairwise logic to a vector set.

Description:

This routine is called by EVALDR to apply Fisher logic which resides at a given node in the logic tree. The logic file is opened, the logic is extracted and the file is closed. The vector set is then opened and the logic is applied to each vector. The vector set is then closed upon completion.

Subroutine Calls:

CLLF, CLOVEC, GETVEC, GTLF, OPLF, OPNVEC

Calling Sequence:

```
JSR       R5,EVALFP
```

Prior to entry, the first nine words of the "FRECOR" buffer must be set to:

Word	Description
0	Logic node number
1	Logic file device (RAD50)
2	Logic file unit number
3,4	Logic file name (RAD50)
5	Vector file device (RAD50)
6	Vector file unit number
7,8	Vector file name (RAD50)
9	Node number at which logic evaluation began (used only by EVALDR)

These values are not modified.

Program Name: EXEC

Purpose:

To control the execution of all functions in the image processing system.

Description:

This routine is the executive control program for the image processing system. It accepts user frame selections and determines the appropriate action to be taken. This could be to display a new frame or to retrieve and execute an option overlay.

The file system programs (FILE1, FILE2, FILE3 and FILE4) are overlays to this program. Therefore, the calls to these routines must be made through EXEC to activate the autoload software of DOS. A portion of this autoload software has been replaced by a portion of EXEC. This change allows overlays from the core resident programs' file and from the option overlay files without confusion. This type of operation is not allowed by the DOS modules that were replaced.

Other functions, such as the automatic log and the error reporting routines are also contained within EXEC. One section of code is exclusively used for initialization functions and is overlayed by the free core buffer following initialization. This area is defined to be in the .PSECT region named "FRECOR". This .PSECT is used to allow the size of the FRECOR buffer to be expanded at link time. A second .PSECT region named "FRESIZ" is included to establish the end of the FRECOR buffer. The reader will note that "FRESIZ" will follow "FRECOR" in core due to the alphabetization performed by the linker. Since the label "ENDCOR" is defined within "FRESIZ" it will then represent the true end of the "FRECOR" buffer.

Subroutines Called:

ALPHA, BLDISP, CONV10, F.C, F.C$, F.CDOS, F.CH, F.CL, F.D, F.E, F.I, F.O, F.P, F.R, F.R$, F.RL, F.RN, F.S, RSTREG, SAVREG, TTYGRF, TTYIN, TTYO, TTYOUT and UNPACK

Calling Sequences:

The main entry point where control is transferred by DOS is at:

E.EXEC

The optional parameter which appears in the four following create and retrieve calling sequences provides the capability for an alternate return should the user respond with nothing other than a carriage return. If this parameter is not specified, an error will be reported for such a condition.

Program Name: EXEC (Continued)

Create Files

```
        JSR     R5,F.CRE$
        BR      End of Param List
        .WORD   Adr. of File Request Block
        .WORD   Adr. of Alternate Return (Optional)
```

NOTE: The buffer specified in the request block is not used for this call.

Create and Open Files

```
        JSR     R5,F.CRE
        BR      End of Param List
        .WORD   Adr. of File Request Block
        .WORD   Adr. of Alternate Return (Optional)
```

Retrieve Files

```
        JSR     R5,F.RET$
        BR      End of Param List
        .WORD   Adr. of File Request Block
        .WORD   Adr. of Alternate Return (Optional)
```

NOTE: The buffer specified in the request block is not used for this call.

Retrieve and Open Files

```
        JSR     R5,F.RET
        BR      End of Param List
        .WORD   Adr. of File Request Block
        .WORD   Adr. of Alternate Return (Optional)
```

Program Name: EXEC (Continued)

Check for File Existence

```
        JSR     R5,F.CHK
        BR      .+6
        .WORD   Adr. of Request Block
        .WORD   Return Adr. if File is Nonexistent
```

NOTE: The buffer specified in the request block is not used for this call.

Access a Record or Group of Contiguous Records

```
        JSR     R5,F.PTR
        BR      .+6
        .WORD   Adr. of File Request Blk.
        .WORD   Return Address if Record Nonexistent
```

Delete a File

```
        JSR     R5,F.DEL
        BR      .+6
        .WORD   Adr. of a Buffer Containing, In Order, The Link (5 WDS.)
                and the File (7 WDS.) Blocks Describing the Desired File
        .WORD   Return Adr. if File is Nonexistent
```

Rename a File

```
        JSR     R5,F.RNM
        BR      .+10
        .WORD   Adr. of a Buffer Containing, In Order, a Link Block
                (5 WDS.), a File Block (7 WDS.) Containing the Current
                Name and a File Block Containing the New Name
        .WORD   Return Adr. if the File is Non-Existent
        .WORD   Return Adr. if a Dup. File Name is Found.
```

Extend a File

```
        JSR     R5,F.EXT
        BR      .+6
        .WORD   Request Blk. Adr.
        .WORD   Adr. of Location Containing Size of Extension in 256 WD Blks.
                the File must be in a Closed State
```

Close a File

```
        JSR     R5,F.CLOS
        BR      .+4
        .WORD   Adr. of File Request Block
```

Close All Open Files

```
        JSR     R5,F.SHUT
```

Initialize a Dataset for Direct DOS Access

```
        JSR     R5,F.INIT
        BR      .+4
        .WORD   Link Blk. Adr.
```

Open a File for Direct DOS Access

```
        JSR     R5,F.OPEN
        BR      .+6
        .WORD   Link Blk. Adr.
        .WORD   File Blk. Adr.
```

Program Name: EXEC (Continued)

Close a File Which Was Open for Direct DOS Access

```
        JSR     R5,F.CLSE
        BR      .+4
        .WORD   Link Blk. Adr.
```

Release a Dataset Which Was Initialized for Direct DOS Access

```
        JSR     R5,F.RLSE
        BR      .+4
        .WORD   Adr. of Link Blk.
```

Request Block Format.

```
        .WORD   Adr. of Char. String for Requesting File Name
        .WORD   File Name 1st WD.
        .WORD   File Name 2nd WD.
        .WORD   File Extension
        .WORD   Device Name in RAD5Ø
        .BYTE   Unit No.
        .BYTE   File Access

                Bit Ø:  Ø - Fixed Length Record
                        1 - Variable Length Record (Not Implemented)

                Bit 1:  Ø - Contiguous File
                        1 - Linked File (Not Implemented)

                Bit 2:  Ø - No Read
                        1 - Read File

                Bit 3:  Ø - No Write
                        1 - Write Into File

        .WORD   Adr. of File Operations Buffer
        .WORD   Buffer Size (BYTES)
```

Program Name: EXEC (Continued)

```
.WORD     Number of Records
.WORD     Record Length
.WORD     No. of Consecutive Rec. Needed in Core Simultaneously
.WORD     Record Number Requested
```

The Error Processing Routine is Called as Follows:

```
JSR       R5,E.ERR
BR        .+4
.BYTE     Error Number
.BYTE     Error Type
```

An overlay can be loaded by setting the appropriate file identification in the .RUN LINK and file blocks (RUNLINK and RUNFIL) and transferring control to "LOADER".

The following three entry points are for routines that support the overlay autoload facility.

```
$RDSEG  -  Read Overlay
$SAVAL  -  Save and Restore Registers
$$WAIT  -  Initialize and Check Overlay Status
```

The following entry point when taken causes the stack to be reset and the current overlay to be removed from memory. Control is then transferred to the Executive.

```
RESTRT
```

Program Name: EXP

Purpose:

To compute the exponential value for a given power.

Description:

This routine raises "e" to the power indicated by the calling argument. The computational method is discussed in Hart[1].

Subroutines Called:

(See calling description.)

Calling Sequence:

```
JSR       R5,EXP
```

The argument is expected to be supplied in ACØ. The result is also returned in ACØ.

Two error returns must be provided which are called as follows:

Positive exponent too large:

```
JSR       R5,EXPBIG
```

Negative exponent too large:

```
JSR       R5,EXPSML
```

If either of these error routines returns via an "RTS PC", an unknown value will be returned in ACØ.

General and floating point registers are modified.

Program Name: FILE1

Purpose:

To create and retrieve data files.

Description:

This program, which is an overlay to EXEC, has five entry points. Two entry points exist for creating files and two exist for retrieving files. One entry point in each case allows the file to be created or retrieved without opening it for access. The other entry point opens the file for access as well. A fifth entry point simply checks for the existence of a file. If it is not found, an alternate return is taken.

Files that are opened have an associated buffer. A header in this buffer is established for accessing the file on disk via the FILE2 overlay. DOS link, file name and tran blocks are established within this header, along with parameters which describe the current contents of the buffer's data area.

A list is maintained within EXEC of all open files. Therefore, as each file is opened, the address of its request block is inserted into this list.

All file operations are performed within the DOS file structure. The DOS programmed requests are used for accomplishing the desired tasks.

Subroutines Called:

E.ERR, GETNAM, RESTRT, R$TREG, SAVREG and TTYIN

Calling Sequence:

Since the routines contained within this program are accessed as an overlay through the executive, detailed calling sequences are provided under EXEC. EXEC transfers control to the following entry points via a "JMP" statement:

F.C$ - Create a file

F.C - Create and open a file

F.R$ - Retrieve a file

F.R - Retrieve and open a file

F.CH - Check for file existence.

Program Name: FILE2

Purpose:

To access data within a file.

Description:

This routine accesses data within files that are opened by the routines in the FILE1 overlay. This routine is also an overlay to the EXEC program.

When a request is made for a record or group of contiguous records, the associated buffer header is examined to determine if the data has been loaded into core by a previous operation. If it is in core, it's address is simply returned to the calling program. If not, the file access mode is checked to determine what operations are to be performed. If the access is "read only", the requested data is read into core. If the access is "write only", the data is written onto the disk and the position for the desired record(s) is established in core. If the access is "read/write", the current buffer contents are written onto the disk and the desired record(s) are then read into core. In all cases where data is read into core, an amount of the data equal to the size of the buffer is read into core. This is done in anticipation of future requests for the next records in sequence. This reduces overall data transfer time.

Subroutines Called:

E.ERR

Calling Sequence:

Since this routine is accessed as an overlay through the executive, a detailed calling sequence is provided under EXEC. EXEC transfers control to the entry point "F.P" via a "JMP" statement.

Program Name: FILE3

Purpose:

To delete, rename and close files and to initialize, open, close and release datasets.

Description:

The delete and rename operations expect DOS link and file name blocks to be prepared as required by the ".DELET" and ".RENAM" programmed requests. These requests are then used to perform the desired function.

Two entry points are available for closing files. The first closes individual files and the second closes all files that are currently open. In either case, the associated file request block addresses are removed from the open file list within EXEC.

To establish control over datasets (as defined under DOS) that have been initialized and/or opened, routines are included herein to maintain lists of such datasets within EXEC. This, for example, prevents an application overlay from initiating a dataset but not releasing it due to a premature termination of the execution of the overlay. These routines provide the initialize, open, close and release dataset functions.

Subroutines Called:

E.ERR, RSTREG and SAVREG

Calling Sequence:

Since the routines contained within this program are accessed as an overlay through the executive, detailed calling sequences are provided under EXEC. EXEC transfers control to the following entry points via a "JMP" statement:

F.D - Delete a file

F.RN - Rename a file

F.CL - Close a file

F.S - Close all open files

F.I - Initialize a dataset

F.O - Open a dataset

F.CDOS - Close a dataset

F.RL - Release a dataset.

Program Name: FILE4

Purpose:

To extend the length of a specified file.

Description:

The file is retrieved and its current size is obtained. The requested extension is then added to this size and a new file is created under the name of "IPS.TMP". Any previous files existing under this name are deleted. The old file is copied into the new file, the old file is deleted and the new file is renamed to that of the old file.

Subroutines Called:

E.ERR, RSTREG and SAVREG

Calling Sequence:

Since this routine is accessed as an overlay through the executive, a detailed calling sequence is provided under EXEC. EXEC transfers control to the entry point "F.E" via a "JMP" statement.

Program Name: FILL

Purpose:

To restore breaks in lines in a binary image.

Description:

This routine scans a binary input image for non-edge points (Ø grey value) whose eight surrounding points meet an "edge, non-edge, edge, non-edge" condition. When this condition is discovered the point is replaced with a value of 255. The result is non-connected edge points separated by one pixel are connected. Two different procedures are followed depending upon which entry point is selected. Either the operation is performed on only data appearing in the input image or on the current state of the union of the input and output images.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT and RAD2AS

Calling Sequence:

Entry if from the Executive at:

START	Regular fill in
START +2	Adaptive fill in

Program Name: FINDPT

Purpose:

To determine the interior points of a region defined by a region file.

Description:

This program determines the points interior to and on a region boundary. A boundary, as opposed to being a line of zero width, is actually one or more pixels wide depending upon the orientation of the line within the array. As a result, this "thick" boundary greatly complicates the task of finding the region points.

The problem is attacked by taking one row at a time and determining its intersection with the boundary lines. These are called intersection intervals. The beginning of a row is assumed to be an exterior point and is therefore a point of reference. When the row reaches an intersection interval the points then are considered interior points. After the interval is passed, the status of the points depends on the interval characteristic. If the intersection interval is not at the end of a boundary line segment, then as the row passes through the interval, it goes from outside the region to inside or vice-versa. Therefore the interval is an "in-out" or "change" interval.

The intervals at the vertices of the line segments are somewhat more complicated. In this case, the characteristic of the interval is "change" if the boundary lines meeting at the vertice approach it from opposite sides of the row in question. If they approach the vertice from the same side, the interval is a "no change" interval. This means that points on the interval are within the region but points on either side of it are either both inside or both outside the region.

Horizontal lines in the boundary are essentially one long intersection interval. Such an interval's characteristic is determined by the lines which meet its end points. A rule similar to the vertice rule above is followed. If the lines at each end approach from opposite sides of the row its characteristic is "change". If they approach from the same side it is "no change". Successive horizontal lines are combined into one long interval.

For each row that crosses the region, the above procedure is followed to determine the interior points. One subroutine (BEGFND) must be called to initialize the operation. Successive calls to FINDPT then return the row and column values found within the region. Each call returns one point.

Another subroutine (SNGLPT) determines if a given point is in the region. The procedure for making this determination is the same as the above.

Program Name: FINDPT (Continued)

Subroutines Called:

E.ERR, F.CLOS, F.PTR, F.RET, RSTFPS, RSTREG, SAVFPS and SAVREG

Calling Sequence:

To retrieve the region file and initialize the working buffer for use by FINDPT and SNGLPT:

```
JSR     R5,BEGFND
BR      .+6
.WORD   Address of the request block for the region file.
.WORD   Address of an alternate return to be taken when the
        user responds to the request for a region file name
        with only a carriage return.
```

Upon return from BEGFND the buffer specified in the request block is still in use even though the region file has been closed. This buffer is used until all calls to FINDPT and SNGLPT have been completed for that region.

To obtain the coordinate of the next point found within the region:

```
JSR     R5,FINDPT
BR      .+6
.WORD   Address of the request block for the desired region
.WORD   Address of an alternate return when there are no
        further points remaining in the region.
```

The coordinate values are returned on the stack with the row on top followed by the column.

To determine if a particular point is within the region:

```
JSR     R5,SNGLPT
BR      .+1Ø.
.WORD   Address of the request block for the desired region
.WORD   Row coordinate value
.WORD   Column coordinate value
```

Program Name: FINDPT (Continued)

.WORD In/Out indicator

Ø = Out

1 = In

Program Name: FISODP

Purpose:

To compute the Fisher direction, the discriminant plane vector and five thresholds.

Description:

This routine computes the above mentioned values for a given class pair. The records in the mean-covariance file (MCFILE.MC) associated with the two classes are passed as parameters. One class pair is treated per call.

Subroutines Called:

CMPRES, EXPAND, E.ERR, FPSQRT, F.PTR, GETIX, INVMAT, LINDEP, MDOTV, RSTREG, SAVREG and SETIXD

Calling Sequence:

Prior to calling "FISHER" the following call must be made:

```
JSR       R5,SETFIS
```

this routine expects the following globals to be provided:

BOTCOR - Address of the beginning of a working buffer

TOPCOR - Address of the end of the working buffer.

The Fisher values are calculated by the following call:

```
JSR       R5,FISHER
```

This routine expects the following globals to be provided:

MCBLK - Address of request block for file MCFILE.MC

MCREC - Record number parameter in request block for MCFILE.MC

MVEC - Buffer containing the measurements from which to create the logic. The first word is the number of measurements and each word thereafter contains a measurement number.

PAGEA - Record number in MCFILE.MC corresponding to the first class.

PAGEB - Record number in MCFILE.MC corresponding to the second class.

Program Name: FISODP (Continued)

VECDIM - Dimension of the vectors.

Upon return, the computed values are found at the addresses given in the following global locations:

DP - Address of discriminant plane vector

FISH - Address of Fisher vector

THRESH - Address of five thresholds.

All values are in single precision floating point. See documentation of logic tree file (Appendix E) for the formats of the return values.

General Registers are not modified.

Program Name: FLTMUL

Purpose:

To form the product between a filter file and a file of another type.

Description:

The product between the files is formed on a point-by-point basis. The second file can have byte, integer, double integer or floating point values. However, it must be an array type of file such as an image or a Hadamard transform of an image.

The output file is created in the same size and format as the second input file. If at any point overflow occurs, the output takes on the largest positive or negative value whichever is appropriate. A count of overflows is maintained and is reported to the user at the end of processing.

Subroutines Called:

DI2DA, E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, GETNAM, PARCOR, RAD2AS, RSTFPS, SAVFPS and TTYOUT

Calling Sequence:

Entry is from the Executive at:

FLTMUL

Program Name: FPASC

Purpose:

To convert from binary floating point to an ASCII character string.

Description:

Two types of conversion are provided, both of which follow FORTRAN conventions. The first is similar to the FORTRAN "F" conversion. That is, the binary value is converted to its equivalent decimal number with a decimal point. The field width and the number of places to the right of the decimal point must be specified.

The second type provides the FORTRAN "E" type conversion. That is, the binary value is converted to a decimal number multiplied by a power of ten. The field width and the number of places to the right of the decimal point must be specified in this case also.

Subroutines Called:

SAVFPS, SAVREG, RSTFPS and RSTREG

Calling Sequence:

Fixed Format Conversion:

```
JSR       R5,FPASCF
BR        .+8.
.BYTE     Width of Field
.BYTE     Number of Digits to the Right of the Decimal Point
.WORD     The Two Word Floating Point Number
.WORD     Address of the Buffer for the Converted Number
```

The field width minus the number of decimal digits must be greater than 1.

Exponential Format Conversion:

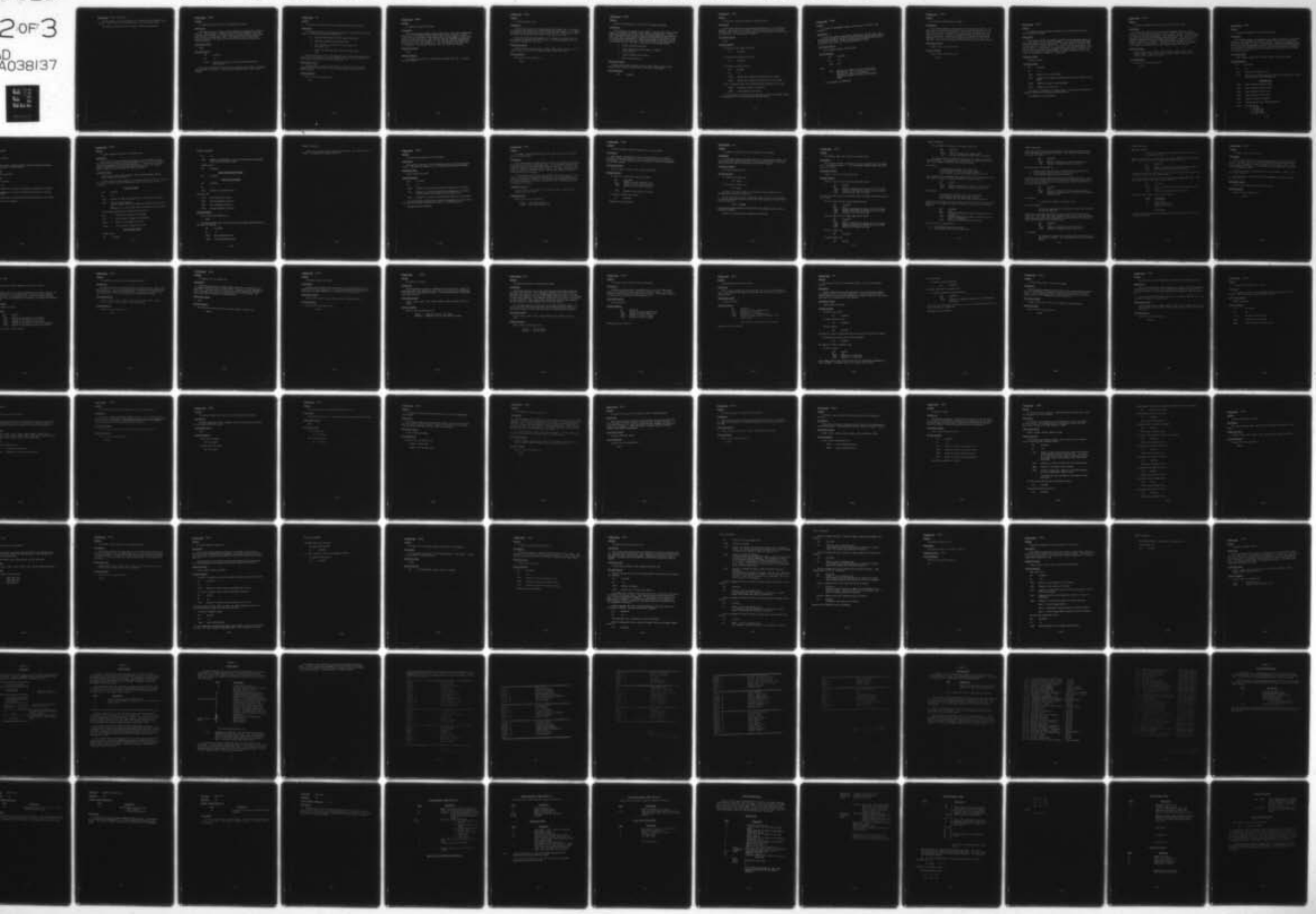
```
JSR       R5,FPASCE
BR        .+8.
```

(The parameter list is identical to that for FPASCF)

The field width minus the number of decimal digits must be greater than 5.

AD-A038 137 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
IMAGE PROCESSING SOFTWARE CONVERSION. COMPUTER PROGRAM DOCUMENT--ETC(U)
FEB 77 J C LIETZ, M J GILLOTTE, D R HOUSE F30602-76-C-0164
UNCLASSIFIED PAR-76-35-VOL-2 RADC-TR-77-51-VOL-2 NL

2 OF 3
AD A038137



Program Name: FPASC (Continued)

In both cases, *'s will be returned in the buffer if a conversion error occurred. The "C" bit is set if this occurs. It is otherwise clear.

The general and floating point register contents are preserved.

Program Name: FPSQRT

Purpose:

To compute the square root of a floating point number.

Description:

The square root of a single or double precision floating point number is computed. The precision is determined from the status register within the floating point processor. An initial guess at the final value is made based upon the half of the exponent value. This results in only three Newton iterations required for single precision and four iterations required for double precision.

Subroutines Called:

None

Calling Sequence:

```
        JSR       R5,FPSQRT

        BR        .+4

        .WORD     Address of the Single or Double Precision Floating
                  Point Argument
```

The result is returned in the subroutine argument and in AC∅. The general registers are not affected. Floating point registers AC∅, AC1 and AC2 are changed.

Program Name: FPW

Purpose:

To control the overall process of creating Fisher pairwise logic.

Description:

This routine monitors the processes required for the creation of Fisher logic. It performs the following functions:

1. Allocate buffer space for the various operations.

2. Retrieve the mean and covariance file.

3. Call "FISHER" to compute the Fisher vectors for each class pair.

4. Insert the resulting logic into the current logic file.

Prior to execution, this routine expects that a logic tree node has been selected for the logic and that the appropriate means and covariances have been computed and stored in the file "MCFILE.MC".

Subroutines Called:

ADDNOD, ADDSYM, CLLOGF, E.ERR, FISHER, F.CLOS, F.PTR, F.RET, GETMEA, GETNOD, GETSYM, GTLOGF, LOADER, OPENLF, OPLOGF, RSTREG, SAVREG, SETFIS, TTYIN, and TTYO

Calling Sequence:

Entry is from the Executive at:

FPW

Program Name: FRAMES

Purpose:

To create the option frame file.

Description:

This routine accepts frame building input from a file called "FRAMES.SRC". Each line in this formatted ASCII file is expected to be either a comment, a blank line, a form feed, a total frame count, a frame header, a frame option entry or a frame "end" line. The lines are read and processed according to type until an "end" line is found. At that time the frame is completed and is written into the file "FRAMES.IPS". The construction of the next frame is then begun. This process continues until all frames are complete.

Subroutines Called:

None

Calling Sequence:

This routine executes as a "stand-alone" program under DOS. The entry point is "FRAMES".

Program Name: GETBOL

Purpose:

To create Boolean logic.

Description:

This routine directs the overall creation of Boolean logic. The logic is entered at the keyboard as a logical/arithmetic expression. It is accepted and compiled into machine code by the routine "CMPILB". Following its entry the user is given the option to reject it and make a new entry.

After the logic has been accepted, it is stored in the logic file and two subnodes are added to the current node. The class assignments for each subnode are obtained from the user.

Subroutines Called:

ADDNOD, ADDSYM, CLLOGF, CLOVEC, CMPILB, E.ERR, GETNOD, GETSYM, GTLOGF, LOADER, OPLOGF, OPNVEC, OPTION, RAD2AS, SI2DA, TTYIN and TTYOUT

Calling Sequence:

Entry is from the Executive at:

GETBOL

Program Name: GETCON

Purpose:

To plot cross sections of a filter on the graphics display.

Description:

This routine is an overlay to the "CREFLT" routine which creates filter files. The function is passed to this routine as machine code. If any undefined constants exist in the code, this routine requests them from the user. The user is also queried to obtain the end points of a line in the X-Y filter plane through which a perpendicular plane is inserted to obtain a cross section of the function. This cross section is then displayed on the graphics display. The user is then given the following options:

1. Accept the filter function;
2. Reject the function and go back to "CREFLT" to get another;
3. Display another cross section;
4. Exit to the executive.

Subroutines Called:

ALPHA, CLEAR, CREFLT, DSABLR, E.ERR, FLTENT, FPASCE, GRMODE, PLOT, RAD2AS, RSTFPS, SAVFPS, SNGDEC, TTYIN, TTYO, TTYOUT and WHICH

Calling Sequence:

```
JSR       R5,GETCON
```

Program Name: GETIX

Purpose:

To retrieve a specified element of a given matrix.

Description:

This routine obtains a matrix element specified by its row and column position. The matrix may be stored in either single or double precision floating point format. As a result, one of two initialization routines must be called prior to requesting any elements.

Subroutines Called:

None

Calling Sequence:

To initialize for single precision:

```
JSR       R5,SETIXF
```

To initialize for double precision:

```
JSR       R5,SETIXD
```

To retrieve a matrix element:

```
JSR       R5,GETIX
BR        .+6
.WORD     Address of a location containing the row number
.WORD     Address of a location containing the column number.
```

Prior to calling GETIX, the following global parameters must be set:

DIMEX - Number of columns in the matrix

MATEX - Core address of the matrix.

The core address of the element is returned in R3 and the element itself is returned in AC3. All other registers are unaffected.

Program Name: GETMEA

Purpose:

To obtain the measurement numbers from the user for Fisher logic creation.

Description:

This routine requests measurement number input from the user. The numbers are accepted as one line of input where the numbers are assumed to be separated by commas. Ranges of measurements are allowed which must be indicated by the limits of the range separated by a dash.

Subroutines Called:

E.ERR, RSTREG, SAVREG, TTYIN and TTYO

Calling Sequence:

```
        JSR     R5,GETMEA
        BR      .+4
        .WORD   ADR
```

where

ADR = Address of a buffer in which to return the measurements. The first word of this buffer contains the number of measurements. Measurement numbers are sorted in increasing order.

All registers are preserved.

Program Name: GTFILE

Purpose:

To create a checkerboard test image.

Description:

This routine creates an image file of 512 rows and 512 columns. The first row is created with grey value Ø in the first 32 columns, 1 in the next 32 and up to 15 in the last 32. This row is then copied into the image file 32 times. The next row sequence begins with value 16 in the last 32 columns and increases by one every 32 colums until reaching the beginning of the row. This row is then copied into rows 33 thru 64 of the file. This alternating process continues until all rows are complete.

Subroutines Called:

E.ERR, F.CRE, F.CLOS and F.PTR

Calling Sequence:

Entry is from the Executive at:

GTFILE

Program Name: HADMAR

Purpose:

To perform a scaled Hadamard transform on a one-dimensional array of double precision integers.

Description:

This routine computes the Hadamard transform in the classical manner of log N steps where N is the number of elements. The sum and difference operations are performed between two buffers. The first buffer is the input array and the second is a working buffer. If N is odd, the result appears in the working buffer. In this case, it is copied back into the array buffer. After the Nth step, the elements are rearranged in sequency order. If during any step of the computation overflow occurs, the array is divided by two. The number of divisions is returned in a scale word.

Subroutine Calls:

RSTREG and SAVREG

Calling Sequence:

```
JSR       R5,HADMAR
BR        .+10
.WORD     Address of a working buffer
.WORD     Address of a location containing the number of elements in the
          array
.WORD     Address of array to be transformed
.WORD     Address of a scale word
```

The result is returned in the input array. The scale word contains the number of times that the array was divided by two.

All registers are not modified.

Program Name: HADXFM

Purpose:

To perform a Hadamard transform on a two-dimensional array.

Description:

This routine applies the one-dimensional Hadamard transform routine (HADMAR) to a two-dimensional array. This is accomplished by first applying it to the rows and then applying it to the resulting columns. The input array can either be an image or a Hadamard transformed image. The output is then a Hadamard transformed image or an image, respectively. When the output is a Hadamard transform, the option is provided to create an image which is a scaled version of the transform. A histogram of the transform is displayed on the graphics display to aid in selecting boundaries for scaling.

Subroutine Calls:

ALPHA, BLDISP, CHCUR, CLEAR, DSABLR, ENABLR, E.ERR, FPASCE, F.CLOS, F.CRE, F.CRE$, F.PTR, F.RET, F.SHUT, GRMODE, HADMAR, PLOT, PAD2AS, RSTREG, SAVREG, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

HADXFM

Program Name: HISPLT

Purpose:

To display histograms on the graphics display.

Description:

This routine displays a histogram on the graphics terminal at a specified location. The horizontal and vertical limits of the plot must be provided along with the number of bins in the histogram. Options are available to draw axes and to draw the histogram as bars or lines or dotted lines connecting the tops of what would normally be the bars. The vertical scaling can optionally be controlled. If not specified, the histogram is plotted to the full range of the space provided.

Subroutine Calls:

ALPHA, DPSQRT, GRMODE, PLOT, RSTFPS, RSTREG, SAVFPS and SAVREG

Calling Sequence:

```
JSR       R5,HISPLT
BR        .+6
.WORD     Address of a parameter list
.WORD     Address of a buffer containing double word histogram bin counts
          (low order part first for each bin)
```

Parameter List

```
.WORD     Upper horizontal limit of plot
.WORD     Lower horizontal limit of plot
.WORD     Upper vertical limit of plot
.WORD     Lower vertical limit of plot
.WORD     Number of bins in histogram
.WORD     Spacing between bins (return parameter)
.WORD     Axis indicator
               0 = No axis
               1 = Y axis only
               2 = X axis only
               3 = X and Y axes
```

.WORD Type of histogram
0 = Bars
1 = Lines
2 = Dashed lines

.WORD Largest double word (low order then high order) value to be represented as full scale. If the first word is -1, then HISPLT uses the largest bin count in the histogram.

.WORD Double word minimum count in histogram (not needed if previous parameter is -1).

All registers are preserved.

Program Name: INVMAT

Purpose:

To invert a matrix.

Description:

This program inverts a square matrix of four-word floating elements. The result is returned in the same buffer.

Subroutine Calls:

GETIX, RSTREG and SAVREG

Calling Sequence:

```
JSR       R5, INVMAT

BR        .+8.

.WORD     Address of a location containing the address of the matrix

.WORD     Address of a location containing the address of a working
          buffer

.WORD     Address of a location containing the dimension of the matrix
```

General registers are not modified.

Program Name: KEYARA

Purpose:

To create a region file based on keyboard input.

Description:

This routine creates a region file based upon user keyboard entries. The entries can be the actual coordinate values or the upper left coordinate value of a rectangular region and its dimensions. A third option to enter cursor coordinates from an image display has been included. The current effort, however, does not include the display software.

Subroutine Calls:

E.ERR, F.CRE, F.PTR, F.RET, F.RET$, F.SHUT, OPTION, RSTREG, SAVREG, SNGDEC, TTYIN, TTYO and TTYOUT

Routines that are not included in the current effort but that are required for cursor coordinate input are expected to have the following calling sequences:

Display an Image

```
JSR       R5,DISP1

BR        .+6

.WORD     Address of image file request block (file must not be open)

.WORD     Blow up factor (integer)
          Positive values blow up the image and negative values shrink
          it.
          Values of -1, 0 and 1 cause every image point to be displayed.
```

The following globals must be set prior to the call:

```
SROW      =    First row in image to be displayed

SCOL      =    First column in image to be displayed

OUTROW    =    First row on display to be used

OUTCOL    =    First column on display to be used
```

Image Display Cursor

Enable Cursor

```
JSR       R5,ENABLC
```

```
BR        .+4

.WORD     Address of subroutine to call when coordinates are captured
          (when cursor interrupt occurs)
```

Disable Cursor

```
JSR       R5,DSABLC
```

Clear Image Display Graphics

```
JSR       R5,CLRGRF
```

Draw Lines on Display

```
JSR       R5,DRWLNE

BR        .+4

.WORD     Address of a parameter list
```

Parameter List:

```
.WORD     Row coordinate of point 1

.WORD     Column coordinate of point 1

.WORD     Row coordinate of point 2

.WORD     Column coordinate of point 2
```

Calling Sequence:

Entry is from the Executive at:

START

The image display cursor routine calls the following subroutine when a coordinate is captured.

```
JSR       R5,TAKCRD

BR        .+6

.WORD     Row coordinate value

.WORD     Column coordinate value
```

Both row and column display values begin at zero, i.e., pixel at row 1, column 1 is displayed at display point 0,0.

Program Name: LINDEP

Purpose:

To find linear dependent rows of a matrix.

Description:

This routine generates a list of dependent rows in a specified square matrix. The matrix is expected to be in four-word floating point format.

Subroutine Calls:

GETIX, RSTREG and SAVREG

Calling Sequence:

```
JSR       R5,LINDEP

BR        .+8

.WORD     Address of a location containing the address of the matrix

.WORD     Address of a location containing the address of a buffer in
          which to return the numbers of the dependent rows

.WORD     Address of a location containing the dimension of the matrix
```

The return buffer consists of the number of dependent rows in the first byte and the numbers of the dependent rows in the remaining bytes.

Registers are not modified.

Program Name: LINES

Purpose:

To create a binary image where only the edges of objects in an input grey level image appear.

Description:

This routine finds the edge points of objects by comparing the averages of arrays of points surrounding each pixel. If the averages exceed a user specified threshold the output binary point is set at 255. If it does not exceed the threshold the output value is zero. The array averages are computed by calling the "SMOOTH" routine. The smoothed image is stored in a temporary file for processing.

A second option called "AREA EDGE DETECTION MAX" builds upon the first option. This routine considers neighboring row and column positions that have also exceeded the threshold. The position that exceeded the threshold by the greatest amount in the row is then set to 255. The same action is performed on the column.

Subroutines Called:

E.ERR, F.CLOS, F.CRE, F.PTR, F.RET, RAD2AS, SI2DA, F.SHUT, SMOOTH, SNGDEC, TTYIN and TTYOUT

Calling Sequence:

Entry is from the Executive at:

LINES - Area edge detection
LINES+2- Area edge detection max.

Program Name: LNPRNT

Purpose:

To print specified character strings on the line printer.

Description:

This routine initializes and opens the line printer output dataset. Character strings terminated by a null are then accepted and listed on the line printer. A second calling point is available to close and release the line printer dataset.

Subroutines Called:

F.CLSE, F.INIT, F.OPEN, F.RLSE, RSTREG and SAVREG

Calling Sequence:

To print one or more ASCII character strings:

```
JSR       R5,LNPRNT
BR        Offset to end of parameter list
.WORD     Address of ASCII character string
.WORD     Address of 2nd string (optional)
          :
          :
.WORD     Address of last string (optional)
```

To close and release the dataset:

```
JSR       R5,LPCLOS
```

Registers are not modified.

Program Name: LOG

Purpose:

To compute the natural log or the common log of a number.

Description:

This routine computes the natural log of a floating point number. The method used is described by Hart.[1] The common log (base 10) is computed from the natural log by multiplying by the constant $Log_{10}(e)$.

Subroutines Called:

(See calling description)

Calling Sequence:

To compute the common log:

```
JSR  R5,LOG
```

To compute the natural log:

```
JSR  R5,LN
```

In both of the above cases the calling and return arguments are in floating point processor register Ø.

If any error occurs in the computation (log of Ø, etc.) this routine will call a programmer supplied routine that must be given the name "LOGERR". The call format is:

```
JSR PC, LOGERR
```

If the error routine returns via an "RTS PC", the original value in ACØ is returned unmodified.

General and floating point registers are modified.

Program Name: LOGGER

Purpose:

To provide a high level access to the logic file.

Description:

This routine contains subroutines to open, close and access the logic file. Included are routines to add, delete and retrieve logic tree nodes and class symbols.

Subroutines Called:

E.ERR, F.CLOS, F.EXT, F.PTR and F.RET

Calling Sequence:

Open the logic file for multiple page read access:

```
JSR       R5,OPLF
BR        .+8
.WORD     Address containing the address of the I/O buffer
.WORD     Address containing the length of the I/O buffer
.WORD     Address of the logic file name
```

Upon return MAXCON contains the maximum number of pages that may be accessed at one time.

Open the logic file for single page read access:

```
JSR       R5,OPENLF
BR        .+8
.WORD     Address containing the address of the I/O buffer
.WORD     Address containing the length of the I/O buffer
.WORD     Address of the logic file name
```

Open the logic file for single page modify access:

```
JSR       R5,OPLOGF
BR        .+8
.WORD     Address containing the address of the I/O buffer
.WORD     Address containing the length of the I/O buffer
.WORD     Address of the logic file name
```

Close the logic file

```
JSR       R5,CLLOGF
```

Close the logic file

```
JSR       R5,CLLF
```

Set a pointer to one page of the opened logic file.

```
JSR       R5,GTLOGF
BR        .+6
.WORD     Address containing the append count
.WORD     Address containing the page number to retrieve
```

The logic file may be opened by using any of the previously defined open subroutines. The append count is the number of pages to add to the file if the requested page does not physically exist.

Upon Return:

```
R3 contains the address of the logic page
FLOGP contains the first page number resident
LLOGP contains the last page number resident plus one
```

Set a pointer to the requested logic pages located in the opened logic file. (The logic file must be opened previously via a call to OPLF.):

```
JSR       R5,GTLF
BR        .+6
.WORD     Address containing the number of pages to read
.WORD     Address containing the starting page number
```

Upon return:

```
R3 contains the address of the first logic page
FLOGP contains the first page number resident
LLOGP contains the last page number resident plus one
```

Add one level of nodes to a lowest node in the logic tree block of the opened logic file. (The logic file must be opened via a call to the subroutine OPLOGF.):

```
JSR       R5,ADDNOD
BR        .+10
.WORD     Address containing node number where nodes are to
          be added
.WORD     Address containing the number of nodes on the new
          level (# of branches)
.WORD     Address to transfer control upon error
```

The error return is executed if:

1. The specified node does not exist
2. The specified node is not a lowest node

Delete one level of logic nodes associated with a superior node from the logic tree block of the opened logic file. (The logic file must be opened via a call to OPLOGF.):

```
JSR       R5,DELNOD
BR        .+6
.WORD     Address containing the superior node number
.WORD     Address to transfer control upon error
```

The error return is executed if:

1. There is more than one level of nodes below the superior node
2. The specified superior node is a lowest node or
3. The specified superior node does not exist.

Retrieve a node from the logic tree block of the opened logic file. (The logic file may be opened via any of the previously defined open subroutines, but if the node information is to be modified, the subroutine OPLOGF must be used to open the file.):

```
JSR       R5,GETNOD
BR        .+6
.WORD     Address containing the node number to retrieve
.WORD     Address to transfer control if the node is inactive.
          (Error Return)
```

Upon Return:

R1 contains the address of the logic node

Upon Error Return:

If R1=0 the logic page is not defined for the node, otherwise the node is inactive.

Retrieve a class symbol entry from the class symbol block of the opened logic file. (The logic file may be opened via any of the previously defined open subroutines, but if the information is to be modified, the subroutine OPLOGF must be used to open the logic file.):

```
JSR       R5,GETSYM
BR        .+6
.WORD     Address containing the node number entry
.WORD     Address to transfer control upon error
```

Upon Return:

R2 contains the address of the second byte in the entry, i.e. the number of symbols. The class symbols follow in successive bytes.

Upon Error Return:

An entry associated with the requested node does not exist in the class symbol block.

Delete a class symbol entry from the class symbol block of the opened logic file. (The logic file must be opened via a call to OPLOGF.):

```
JSR       R5,DELSYM
BR        .+6
.WORD     Address containing the node number of the entry
.WORD     Address to transfer control upon error
```

An error occurs if the subroutine fails to find an entry associated with the node number located in the parameter list.

Add a symbol entry to the class symbol block of the opened logic file. (The logic file must be opened via a call to OPLOGF.):

```
JSR       R5,ADDSYM
BR        .+6
.WORD     Address containing the address of the class symbol
          entry
.WORD     Address to transfer control upon error
```

The class symbol entry is described as follows:

Byte	Description
Ø	Node number
1	Number of symbols
2	First symbol (ASCII)
	:
	:
	Last Symbol

An error occurs if there already exists an entry in the class symbol logic block for the node.

Program Name: LOGOVR

Purpose:

To start, stop or dump the log.

Description:

This routine controls the operation of the automatic log routine found in EXEC. To start the log the log file LOGFIL.IPS is created as a linked file. If the file exists the user is asked if it should be listed. It is then deleted and recreated. The log routine is then enabled by setting bit Ø of the parameter "LOGFLG".

The log is stopped by clearing bit Ø of the parameter "LOGFLG". The file is not deleted.

The log file can be dumped at any time regardless of whether the log is active or not.

Subroutines Called:

E.ERR, F.DEL, SNGDEC, TTYIN, TTYIO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

LOGOVR

Program Name: LPHDCP

Purpose:

To list grey value or binary images on the line printer.

Description:

This routine lists grey value or binary images on the line printer. A three digit number is printed for each pixel value within the user selected portion of the image. For binary images binary 1 is printed as an asterisk and a binary Ø as a blank. If the number of columns is greater than the width of the line printer, the output is generated in several sections.

Subroutines Called:

E.ERR, F.CLOS, F.PTR, F.RET, LNPRNT, LPCLOS, RAD2AS, SI2DA, SI2DAR, SNGDEC, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

EDGPRN - List binary image
GRYDMP - List grey value image

Program Name: MDOTV

Purpose:

To compute the dot product between a matrix and a vector.

Description:

This routine forms the dot product between a matrix and a vector. The elements of each must be in floating point. Single or double precision values are accepted. The processor must be set to the desired precision prior to entry. The column dimension of the matrix must equal the dimension of the vector.

Subroutines Called:

GETIX, RSTREG and SAVREG

Calling Sequence:

```
JSR       R5,MDOTV
BR        .+12
.WORD     Address of the address of the matrix
.WORD     Address of the address of the vector
.WORD     Address of the address for the result
.WORD     Address of the number of columns in the matrix
.WORD     Address of the number of rows in the matrix
```

Registers AC0, AC1 and AC3 are modified.

Program Name: MENCOV

Purpose:

To compute the mean vector and covariance matrix.

Description:

The mean vector and covariance matrix is computed for each class in the current data set. Only those vectors which are located at the current logic tree node will be used for the caluclations. The output is stored in the file "MCFILE.MC". If the file exists, it is deleted and recreated.

Subroutines Called:

CLLOGF, CLOVEC, E.ERR, F.CLOS, F.CRE, F.DEL, F.PTR, GETSYM, GETVEC, LOADER, OPENLF, OPNVEC, RSTREG, SAVREG and SNGDEC.

Calling Sequence:

Entry is from the Executive at:

MENCOV

Program Name: MESAGE

Purpose:

To generate the file ERRORS.IPS.

Description:

This routine converts the error message format of file ERRORS.SRC into the system acceptable format of ERRORS.IPS. Two passes are made through the input file. One pass counts the error types and the messages within each type to determine the size of the output file which is contiguous. The second pass then accomplishes the reformatting.

Subroutines Called:

None

Calling Sequence:

This routine is executed via the DOS RUN command. Entry is at:

MESAGE

Program Name: MODREP

Purpose:

To remove image noise points.

Description:

This routine determines the modal value of the pixels within a 3 by 3 neighborhood of each image point. If the center point differs in value by more than a specified amount, it is replaced with the modal value.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, RAD2AS, RSTREG and SAVREG

Calling Sequence:

Entry is from the Executive at:

MODREP

Program Name: NORMLZ

Purpose:

To normalize an image.

Description:

The normalization process is performed in one of two ways. Either the image grey values are expanded to the full Ø to 255 range or the values are normalized over a user specified range. The function performed is dependent upon the entry point.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, RAD2AS, RSTREG, SAVREG, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

NORMLZ - Normalize over Ø - 255 range
NORMLZ+2 - Normalize over the user specified range.

Program Name: ODDLD

Purpose:

To remove noise points or lines from an image.

Description:

This routine performs one of the two noise removal functions based upon the entry point selected. Both operations, however, make their decisions based upon the state of the 3 by 3 neighborhood about each pixel. The odd dot entry point searches for pixels that differ from one or more or the average of their eight neighbors. The "differ" criterion and the number of neighbors are user specified. Those pixels which satisfy the condition are replaced with the average of those neighbors with which they differ.

The odd line entry point functions in a slightly different manner. The point is replaced with the average only if two neighbors which themselves are adjacent cannot be found that are within the user-specified threshold.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, RAD2AS, SI2DA, SNGDEC, TTYIN and TTYO

Calling Sequence:

Entry is from the Executive at:

ODDLIN - Odd line entry
ODDLIN+2 - Odd dot entry

Program Name: OPTION

Purpose:

To display option lists and accept selections.

Description:

This routine simply lists a specified option list (not frame option lists) on the graphics display and accepts the user's input. The return address is then selected on the basis of this input. The number of options is determined by the length of the parameter list.

Subroutines Called:

E.ERR and SNGDEC

Calling Sequence:

```
JSR       R5,OPTION
BR        Offset to end of parameter list
.WORD     Address of output option list
.WORD     Address of option 1 return
.WORD     Address of option 2 return
          :
          :
```

Registers are not modified

Program Name: PARCOR

Purpose:

To partition the free core buffer.

Description:

This routine divides the buffer "FRECOR" into a number of fractional parts. The addresses and sizes of these parts are then placed directly into request blocks.

Subroutines Called:

SAVREG and RSTREG

Calling Sequence:

```
JSR       R5,PARCOR
BR        Offset to end of parameter list
.WORD     Denominator of fraction
.WORD     Numerator of 1st fractional part
.WORD     Address of buffer address parameter in file
          request block
          :
          :
          (any number of additional 2 word entries).
```

Registers are not modified.

Program Name: PLOT

Purpose:

To provide routines for generating graphic plots on the Tektronix display.

Description:

Several subroutines are contained within this program which together allow graphic plots to be easily generated. A special character string output routine is also provided because the DOS driver does not allow control characters to be transferred to the terminal.

Subroutines Called:

E.ERR, RSTREG and SAVREG

Calling Sequence:

To select alpha mode:

```
JSR     R5,ALPHA
```

To select graphics mode:

```
JSR     R5,GRMODE
```

To clear screen:

```
JSR     R5,CLEAR
```

The display is left in alpha mode with the cursor at the upper left margin.

To position the cursor at the "home" position:

```
JSR     R5,HOME
```

The display is left in graphics mode.

To plot a vector:

```
JSR     R5,PLOT
BR      .+6
.WORD   Address of X coordinate
.WORD   Address of Y coordinate
```

This routine draws a dark vector if the call is immediately preceeded by a call to GRMODE. Successive calls to PLOT draw light vectors.

PLOT (Continued)

To output a single character:

```
        JSR     R5,OUTCAR
```

The ASCII character must appear in R4.

To output a graphics character string:

```
        JSR     R5,TTYGRF
        BR      .+4
        .WORD   Address of a character string which is terminated
                by a null.
```

After this call the global "TTYADR" is set to the next byte address following the terminating null character.

Registers are not modified.

Program Name: PTEDGE

Purpose:

To find object edges in grey value images.

Description:

This routine is designed to find points which are the edges of objects. This is accomplished by comparing each point to user specified neighbor points. If the difference exceeds a user specified threshold, the output point is set to 255. Otherwise the output value is Ø.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, RAD2AS, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

PTEDGE

Program Name: PWLINR

Purpose:

To accept and apply user-specified piecewise-linear transfer functions.

Description:

This routine allows the user to specify the transfer function by entering coordinates at the keyboard. These coordinates describe the end points of the linear sections. The transfer function can be saved in a file for future use.

The newly specified transfer function or a previously existing one can be applied to an image via this routine.

Subroutines Called:

ALPHA, BLDISP, CLEAR, DSABLR, ENABLR, E.ERR, F.CRE, F.RET, F.PTR, F.SHUT, GRMODE, OPTION, OUTCAR, PARCOR, PLOT, RAD2AS, SNGDEC, TRNFTN, TTYIN, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

PWLINR

Program Name: RAD2AS

Purpose:

To convert RAD50 characters to ASCII

Description:

This routine converts a string of RAD50 packed characters to an ASCII string. The DOS programmed request ".RADUP" is used.

Subroutines Called:

RSTREG and SAVREG

Calling Sequence:

```
JSR         RS, RAD2AS
BR          .+8
.WORD       Address of RAD50 string
.WORD       Address for ASCII output
.WORD       Number of words in RAD50 string
```

Program Name: RENAME

Purpose:

To rename a file

Description:

This routine allows a file to be renamed. The old file name and new name are obtained from the user. No change in the extension is allowed.

Subroutines Called:

E.ERR, F.RNM and GETNAM

Calling Sequence:

Entry is from the Executive at:

RENAME

Program Name: RGSTAT

Purpose:

To compute and list region statistics.

Description:

This routine computes and lists on the graphics display or line printer the mean, standard deviation, mode, and population of a selected region of an image file or sets of regions and images as described by a spectral set file.

Subroutines Called:

BEGFND, BLDISP, CFREQ, CSDEV, CLEAR, DSABLR, ENABLR, E.ERR, F.CLOS, F.PTR, F.RET, F.SHUT, FINDPT, GMEAN, GSDEV, GMEDN, GMODE, LNPRNT, RAD2AS, RSTREG, SAVREG, SI2DA, SMINIT, SNGDEC, TTYIN, and TTYO.

Calling Sequence:

Entry is from the Executive at:

RGSTAT - Operate on spectral set

RGSTAT+2 - Operate on single region/image pair

Program Name: RTIOMG

Purpose:

To compute the ratios of two images on a point by point basis.

Description:

This routine makes two passes through the file. On the first pass, the range of values is obtained by multiplying each image point to be normalized by 256 and dividing the result by the reference image point. On the second pass, these values are then normalized between 0 and 255.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, PARCOR, RAD2AS, and TTYOUT

Calling Sequence:

Entry is from the Executive at:

RTIOMG

Program Name: SARFPS

Purpose:

To save and restore the status of the floating point processor.

Description:

The FPP's registers, status register, and error trap vector are saved on and restored from the stack:

Subroutines Called:

None

Calling Sequence:

To save the FPP status:

```
JSR  R5, SAVFPS
```

To restore the FPP status:

```
JSR  R5, RSTFPS
```

Program Name: SAVER

Purpose:

To save and restore the general registers R0 - R5.

Description:

The general registers R0 - R5 are saved on and restored from the stack.

Subroutines Called:

None

Calling Sequence:

To save registers:

```
JSR  R5, SAVREG
```

To restore registers:

```
JSR  R5, RSTREG
```

Program Name: SELNOD

Purpose:

To obtain the node number from the user and to call the appropriate logic routine.

Description:

This routine requests the desired node number from the user and stores it in the global location "LOGNOD". Control is then passed to the logic creation overlay associated with the entry point chosen.

Subroutines Called:

E.ERR, LOADER, and SNGDEC

Calling Sequence:

Entry is from the Executive at:

SELNOD - Fisher logic

SELNOD + 2 - Boolean logic

Program Name: SELOGF

Purpose:

To retrieve or create a logic file.

Description:

The existence of the current vector set is checked as described by global "VECTNM." A fatal error is reported if none exists. The logic file name is requested next. If the file exists, the class symbols in the vector file set and the logic file are compared to insure compatibility. The vectors are all reset to node 1 in the logic tree. The logic evaluation overlay is then called to apply any existing logic to the vectors.

If the file does not exist, a new one is created. The class symbols are extracted from the vector file and placed in the logic file.

Subroutines Called:

ADDSYM, CLLOGF, CLOVEC, CRLOG, E.ERR, F.CHK, GETNAM, GETSYM, GETVEC, GTLOGF, LOADER, OPENLF, OPNVEC, RSTREG, SAVREG, and TTYOUT.

Calling Sequence:

Entry is from the Executive at:

START

Program Name: SELVEC

Purpose:

To obtain the vector set file name for future logic operations.

Description:

This routine requests the vector set file name from the user and stores it in the global variable "VECTNM". The vector set is opened to obtain the vector dimension, the vector measurement format indicator and the vector header format indicator. These are stored in global locations "VECDIM", "VECMFI" and "VECHFI" respectively.

Subroutines Called:

GETNAM, OPNVEC and CLOVEC

Calling Sequence:

Entry is from the Executive at:

SELVEC

Program Name: SHIFTI

Purpose:

To shift an image vertically and horizontally.

Description:

This routine shifts an image by a user specified row and column amount. The shift is accomplished by a circular rotation of the rows and columns in the image.

Subroutines Called:

E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, RAD2AS, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

SHIFTI

Program Name: SIMSMO

Purpose:

To obtain a user specified box size and then call the smoothing routine.

Description:

This routine obtains the input and output files and the dimensions of the smoothing array. These parameters are then passed to program "SMOOTH".

Subroutines Called:

BYTDEC, F.CRE, F.RET, F.SHUT, RAD2AS, SI2DA, SMOOTH and TTYOUT

Calling Sequence:

Entry is from the Executive at:

SIMSMO - M by N smoothing array

BOXSMO - Square smoothing array

Program Name: SMOOTH

Purpose:

To smooth an image.

Description:

The smoothing operation is performed by computing the average value of the points in a specified neighborhood about each point and replacing the center point with this value. Edge points are effectively extended outward to satisfy "missing" neighborhood points for edge points.

Subroutines Called:

E.ERR, F.PTR, RSTREG and SAVREG

Calling Sequence:

```
        JSR       R5,SMOOTH
        BR        .+8
        .WORD     Address of input file request block
        .WORD     Address of output file request block
        .BYTE     Number of rows in smoothing array
        .BYTE     Number of columns in smoothing array
```

The files are assumed to be open.

Program Name: SPAMEZ

Purpose:

To compute the mean, variance, standard deviation, median, mode, high, low and range of a set of numbers.

Description:

The statistics are computed on a frequency table that is constructed by this routine. The frequency table is the number of times that each grey value occurred in the data passed to "CFREQ".

Subroutines Called:

RSTFPS, RSTREG, SAVREG, SAVFPS and SQRT

Calling Sequence:

To initialize the frequency buffer, the sum and the sum of squares prior to calling CFREQ, CMEAN and CVARI:

```
JSR     R5,SMINIT
BR      .+8.
.WORD   Address of data value (byte value at an even location)
        to be supplied by CFREQ, CMEAN or CVARI.  Also this
        is the address where the computed values are returned
        from GMEAN, GVARI, GSDEV, GMEDN, GMODE, GHIGH, GLOW
        and GRANGE
.WORD   Address of a 256 word buffer for the frequency table.
.WORD   Address of the number of data values.
.WORD   Address of mode flag.  This is an optional parameter.
        It must be supplied if GMODE is used.

        The mode flag is set by GMODE to the number of modes
        minus one.
```

To tally a byte of data into the frequency table:

```
JSR     R5,CFREQ
```

To add data to the current sum:

```
JSR     R5,CMEAN
```

To add data to the sum and the square of data to the sum of squares:

```
JSR       R5,CVARI (or CSDEV)
```

To calculate the mean of the data:

```
JSR       R5,GMEAN
```

CMEAN must be called first

To calculate the variance of the data:

```
JSR       R5,GVARI
```

CVARI must be called first.

To calculate the standard deviation of the data:

```
JSR       R5,GSDEV
```

CVARI must be called first.

To calculate the median of the data:

```
JSR       R5,GMEDN
```

CFREQ must be called first.

To calculate the mode of the data:

```
JSR       R5,GMODE
```

CFREQ must be called first.

To calculate the high data value:

```
JSR       R5,HIGH
```

CFREQ must be called first.

To calculate the low data value:

```
JSR       R5,LOW
```

CFREQ must be called first.

To calculate the range of values:

```
JSR       R5,RANGE
```

CFREQ must be called first.

Program Name: SPBIAS

Purpose:

To add a constant to an image.

Description:

This routine simply adds a user specified constant to each point in an image.

Subroutine Calls:

DI2DAR, F.CRE, F.PTR, F.RET, F.SHUT, RAD2AS, SI2DA, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

SPBIAS

Program Name: SQRT

Purpose:

To compute the square root of a number.

Description:

This routine computes the square root of a double or single precision integer. The value is computed using Newton's method with an initial guess of (2**15)-1.

Subroutine Calls:

RSTREG and SAVREG

Calling Sequence:

Double precision argument:

```
JSR       R5,DPSQRT
BR        .+4
.WORD     Address of the double precision integer (low order part first)
```

The square root is returned in the low order word of the argument.

Single precision argument:

```
JSR       R5,SQRT
BR        .+4
.WORD     Address of single word integer
```

The square root is returned in place of the argument.

Registers are not modified.

Program Name: TAPE

Purpose:

To perform various tape operations.

Description:

This routine reads and writes image files from and to magnetic tape. The images are stored on tape as one row per record. The last record is followed by an end-of-file mark.

A skip file option and a rewind option are also available.

Subroutine Calls:

E.ERR, F.CLOS, F.CRE, F.INIT, F.PTR, F.RET, F.RLSE, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

ENTRY	-	Read tape file
ENTRY+2	-	Write tape file
ENTRY+4	-	Skip tape file
ENTRY+6	-	Rewind tape

Program Name: TEKDSP

Purpose:

To display a binary image on the Tektronix display.

Description:

This routine displays a binary image on the Tektronix where binary "one" points are displayed as a dot (period) and binary "zero" points are displayed as a blank position. The user can specify that an array of dots be used for a point and he can also specify the spacing between dots. Both have a "blow up" effect.

Subroutine Calls:

ALPHA, BLDISP, CLEAR, E.ERR, F.PTR, F.RET, F.SHUT, GRMODE, HOME, PLOT, RAD2AS, SI2DA, SNGDEC, TTYIN, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

TEKDSP

Program Name: TELEIO

Purpose:

To provide graphics display I/O.

Description:

This routine allows character strings to be printed on the graphics display and to be accepted from the keyboard. Two output routines are included. One appends a carriage return and line feed to the string, and the other does not.

The rebuilding of the option frames is also controlled by this routine. Two calling sequences are provided to enable and disable this function.

Subroutine Calls:

E.ERR, LOGGER, RSTREG and SAVREG

Calling Sequence:

To output a character string with appended carriage return and line feed:

```
JSR       R5,TTYOUT
BR        .+4
.WORD     Address of output string (terminated with a null)
```

To output a character string without appending characters:

```
JSR       R5,TTYO
BR        .+4
.WORD     Address of output string (terminated with a null)
```

Following calls to either TTYOUT or TTYO, the global parameter "TTYADR" is set to the address of the first byte after the null.

To input a character string

```
JSR       R5,TTYIN
BR        .+4
.WORD     Input buffer address
```

The word immediately preceeding the buffer must contain the size of the buffer in bytes. The input string is terminated with a null in place of the final

carriage return and line feed.

To enable frame rebuild:

```
JSR       R5,ENABLR
```

The line counter is reset to the maximum (LINMAX).

To disable frame rebuild:

```
JSR       R5,DSABLR
```

Program Name: TMPARA

Purpose:

To report that the image display options are not implemented.

Description:

This routine simply prints "OPTION INOPERATIVE!!!" when called. Control is then returned to the executive.

Subroutine Calls:

TTYOUT

Calling Sequence:

```
JSR     R5,DISP1'(ENABLC, DSABLC, CLRGRF or DRWLNE)
```

Program Name: TRNFTN

Purpose:

To apply an image transfer function.

Description:

This routine applies a specified transfer function to an image. This function is in the form of a 256 byte look-up table. The input image grey value is used as an index into this table to determine the output value.

Subroutine Calls:

F.PTR, RSTREG and SAVREG

Calling Sequence:

```
JSR       R5,TRNFTN
BR        .+8.
.WORD     Address of input file request block
.WORD     Address of output file request block
.WORD     Address of transfer function in memory.
```

Registers are not modified.

Program Name: TTYIO

Purpose:

To provide high level terminal I/O.

Description:

This routine provides terminal I/O functions that request parameters from the user and accept his response. These parameters include file specifications and numerical input (octal and decimal). Floating point numerical input is accepted. Routines also are provided to convert ASCII character strings to binary.

Subroutine Calls:

E.ERR, RSTFPS, RSTREG, SAVFPS, SAVREG, TTYIN and TTYO

Calling Sequence:

Output a message and input a file specification (the file name is checked for legality)

```
JSR       R5,GETNAM
BR        .+6
.WORD     Address of message
.WORD     Address of a 12 word return buffer
```

This buffer will contain a link block in the lst 5 words and a file block in the last 7 words upon return. Link and file blocks are as described in the DOS monitor programmer's handbook. If the device or unit number is not given, zero is returned. The "wild card" specification is not altered by this routine (52 octal is passed to the calling routine as explained in the DOS programmer's handbook).

Output a message and input a file specification (the file name is not checked for legality. Only the device is of interest).

```
JSR       R5,GETDEV
BR        .+6
```

(The parameter list is identical to that for GETNAM.)

Output a message and input a string of signed double word integer decimal numbers.

```
JSR       R5,DBLDEC
```

```
BR        Offset to end of parameter list

.WORD     Address of message

.WORD     Address of location containing the maximum count of numbers to
          input.  This count must be satisfied exactly or an error message
          is printed and a request is made to retype the line.

.WORD     Address of array for numbers
          This array must contain sufficient space to store all the numbers
          indicated by the second parameter.  The numbers are entered
          on the keyboard separated by commas and preceeded by a minus
          or plus sign.  The absence of a sign is interpreted as a plus
          sign.  Zero is returned for a given number if its corresponding
          field is empty (comma only).

.WORD     Address of location containing CTRL-Z indicator (optional
          argument)
          When specified this argument indicates that the first character
          entered is to be compared to a CTRL-Z.  If the character is not
          a CTRL-Z, a zero will be returned in the indicator.  If it is a
          CTRL-Z, then a -1 is returned.
```

Output a message and input a string of signed single word integer decimal numbers.

```
JSR       R5,SNGDEC

BR        Offset to end of parameter list
          (The parameter list and description is indetical to DBLDEC
          except that single word values are returned.)
```

Output a message and input a string of signed single byte integer decimal numbers.

```
JSR       R5,BYTDEC

BR        Offset to end of parameter list
          (The parameter list and description is identical to DBLDEC
          except that single byte values are returned.)
```

Output a message and input a string of signed double word integer octal numbers

```
JSR       R5,DBLOCT

BR        Offset to end of parameter list
          (The parameter list and description is identical to DBLDEC.)
```

Output a message and input a string of signed single word integer octal numbers.

```
JSR     R5,SNGOCT

BR      Offset to end of parameter list
        (The parameter list and description is identical to DBLDEC
        except that single word values are returned.)
```

Output a message and input a string of signed single byte integer octal numbers.

```
JSR     R5,BYTOCT

BR      Offset to end of parameter list
        (The parameter list and description is identical to DBLDEC
        except that single byte values are returned.)
```

Output a message and input a floating point string of numbers. These numbers can be in "F" or "E" format.

```
JSR     R5,FLTENT
BR      Offset to end of parameter list
        (The parameter list and description is identical to DBLDEC
        except single precision floating point values are returned.)
```

Convert a signed decimal ASCII character string to binary.

```
JSR     R5,CONV10
        Prior to entry, R2 should be loaded with the address of the
        character string.  The binary value will be returned in R1.  A
        null or a comma terminates the string.
```

Convert a signed octal ASCII character string to binary.

```
JSR     R5,CONV8
        (Parameters are passes as for CONV10.)
```

General and FPP registers are not modified.

Program Name: UTILTY

Purpose:

To provide an exit to DOS.

Description:

This routine exits to DOS via an "EMT 60".

Subroutine Calls: None

Calling Sequence:

Entry is from the Executive at:

DOS

Program Name: VECFNS

Purpose:

To provide sequential access to vectors in a vector set.

Description:

This routine provides services to open a set of vector files listed in a vector set file, to access the files and to close the files. The access is on a vector by vector basis. The vectors are returned sequentially. Only one vector file is open at a time.

Subroutine Calls:

E.ERR, F.CLOS, F.PTR, F.RET, F.RET$, RSTREG and SAVREG

Calling Sequence:

Open vector set:

```
JSR     R5,OPNVEC
BR      .+12.
.WORD   Address of the address of an I/O buffer
.WORD   Address of the length of the buffer
.WORD   Address of a four word vector set file specification (device,
        unit, and filename)
.WORD   Address of location containing open access (0 = read, 1 =
        read/write)
.WORD   Address of a three word buffer for return parameters.
        Word 0 = Vector dimensionality
        Word 1 = Measurement format indicator (from file header)
        Word 2 = Vector header format indicator (from file header)
```

Retrieve next sequential vector:

```
JSR     R5,GETVEC
BR      .+4
.WORD   Return address when no further vectors remain
```

The memory address of the vector is returned in R0.

Close vector set:

```
JSR        R5,CLOVEC
```

Program Name: WTDSMO

Purpose:

To perform a weighted smooth.

Description:

This routine performs the smooth by operating on a user specified neighborhood of each point. Each neighbor is multiplied by a weight which is stored in a weight array. This array is equal in dimension to that of the neighborhood. The sum of the products is then divided by the sum of the weights, the sum of the weight absolute values or a user-entered number. The form of this divisor is specified by the user.

Subroutine Calls:

BYTDEC, E.ERR, F.CRE, F.PTR, F.RET, F.SHUT, OPTION, RAD2AS, SI2DAR, SNGDEC, TTYIN, TTYO and TTYOUT

Calling Sequence:

Entry is from the Executive at:

WTSMO	-	Weighted smooth
ABSWS	-	Absolute value weighted smooth

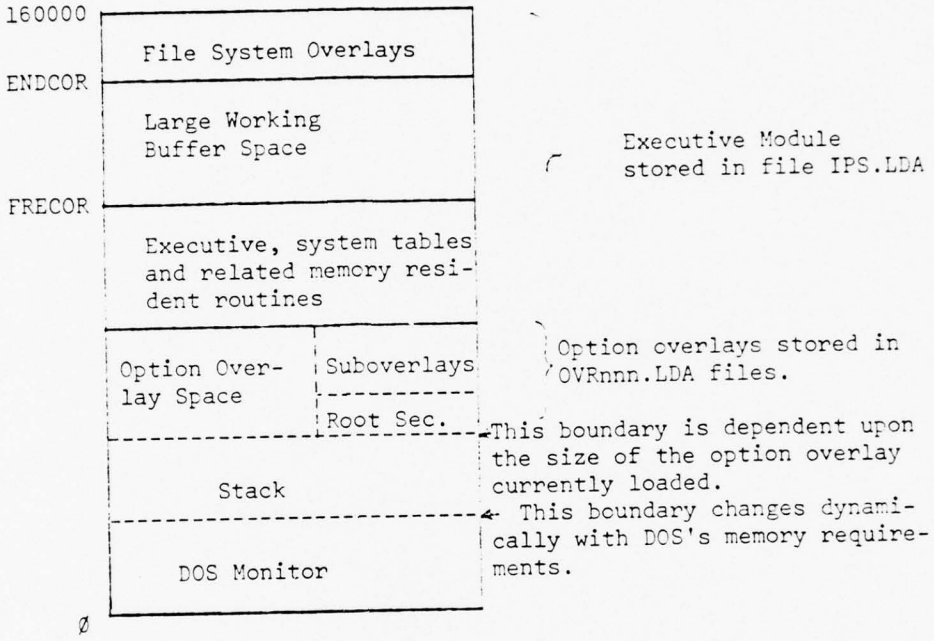
REFERENCES

1. Hart, J.F., et al., Computer Approximations, John Wiley & Sons, 1968.

APPENDIX A

MEMORY MAP

The following diagram is a map of memory when the image processing system is loaded and executing. Specific addresses are not provided since they may vary depending upon the parameters used when building the system.



APPENDIX B

SYSTEM TABLES

A number of system tables exist which are used to store the current operating state of the system. With one exception, they deal with the opening and closing of data files and DOS data sets. The exception is concerned with the option entries associated with the frame being displayed. Due to the specialized nature of this table, it is described in Appendix C along with the frames and the frame file.

The first table deals with the files that have been opened by file calls F.CRE and F.RET. As a file is opened, its associated request block address is stored in the first non-used slot. The table is located in program EXEC at label "FSLOTS". Its format is as follows:

Word	Description
0	Number of slots in table (currently 10).
1	Request block address of an open file (0 if slot is empty).
2	Same as word 1
⋮	⋮

As the files are closed, their associated entries are removed from the table.

Another table is used to store the list of data sets that have been ".INIT'ed" by DOS as requested via the F.INIT file call. This table is also found in program EXEC, but under the label "ISLOTS". This table has the same format as FSLOTS except that the link block addresses are stored and the number of slots in the table is set at five. The entries are removed when their associated data sets are released via the file call F.RLSE.

The last table is used to store the list of files that have been "OPENed" by DOS. Again, this table is found in EXEC under the label "OSLOTS". The format of this table is also the same as FSLOTS except that the associated link block addresses are stored and the number of slots in the table is set at five. The entries are removed when their associated files are closed via the file call F.CLSE.

If it becomes necessary to expand any of the above tables, program EXEC must be modified. This is accomplished by changing the first word of the affected table to reflect the number of required slots and then expanding the buffer space assigned to the table. After EXEC has been re-assembled, the entire system must be rebuilt.

APPENDIX C

SYSTEM FRAMES

The frames actually form a system table where each frame is one page of the table. The frames reside on disk in a contiguous file and are paged into memory (buffer FRMTBL in program EXEC) as required by the system. Each page consists of a maximum of 256 words and appears in the file in the following format:

	Word	Description
	Ø	Frame number in binary
	1	Frame Ø: The total number of frames All other frames: Not used
	2	Reserved for future use
First Option Entry	3	If the related option references another frame this contains the characters "FRM" packed into RAD5Ø format. If the option references an overlay, this contains the last three characters of the overlay name.
	4	The lower byte contains the frame number if the option references a frame, or the overlay entry point number if the option references an overlay. The first entry point is Ø. The upper byte is reserved for future system use.
	5	Reserved for future system use.
Second Option Entry	6	Same as word 3
	7	Same as word 4
	8	Same as word 5
	⋮	⋮
Last Option Entry (Max. of 20)		
	64 - 255	Beginning of frame text. One character string appears here for each option entry above. The first character string is the frame header. Each string is terminated by a "$" except the last string which is terminated by "$$". All text is packed into RAD5Ø format.

The frames are stored in consecutive order in the contiguous file "FRAMES.IPS". If the logical block size of the device is less than 256 words, then several blocks are used to store each frame. If the logical block size is greater than 256 words, then only one frame is stored per block. In this case the remaining words in each block are not used.

The frames are not specified by the user in the format discussed above. Instead, a more easily managed file called FRAMES.SRC is created. This file is in formatted ASCII form and its current contents are listed on the following pages. Consult Section 3 for frame changes.

```
$*************************************************************************
$************************ IMAGE PROCESSING SYSTEM FRAMES ********
$*************************************************************************
$TOTAL NO. OF FRAMES:
14
$*************************************************************************
FRM000                                  MASTER FRAME
TRNFRM   1                              DATA EDITING
TRNFRM   2                              TRANSFER FUNCTIONS
TRNFRM   3                              STATISTICS
TRNFRM   4                              SMOOTH IMAGES
TRNFRM   5                              COMBINE IMAGES
TRNFRM   6                              NOISE REDUCTION
TRNFRM   7                              EDGE DETECTION
TRNFRM   8                              TRANSFORMS
TRNFRM   9                              FEATURE EXTRACTION
TRNFRM   10                             LOGIC DESIGN
TRNFRM   11                             FILE MANIPULATIONS
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   13                             UTILITY FUNCTIONS
END
$*************************************************************************
FRM001                                  DATA EDITING
OVR002   0                              MULT AREA SELECT
OVR036   0                              SPECIFY REGION COORDS
OVR009   0                              SHIFT IMAGE
OVR011   0                              DECIMATION
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
$*************************************************************************
FRM002                                  TRANSFER FUNCTIONS
OVR004   0                              ADD CONSTANT
OVR007   1                              THRESHOLD
OVR008   0                              NORMALIZE
OVR007   0                              ELEMENT CHANGE
OVR008   1                              RANGE CHANGE
OVR020   0                              ARBITRARY FUNCTION
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
$*************************************************************************
```

```
;*************************************************************************
FRM003                                  STATISTICS
OVR005   0                              IMAGE HISTOGRAMS
OVR005   1                              REGION HISTOGRAMS
OVR047   0                              REGION STATISTICS
OVR047   1                              MULTIREGION STATISTICS
OVR005   2                              IMAGE ROW INTENSITY PROFILE
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
;*************************************************************************
FRM004                                  SMOOTH IMAGES
OVR021   1                              BOX SMOOTH
OVR021   0                              M X N SMOOTH
OVR022   0                              WEIGHTED SMOOTH
OVR022   1                              WGHTD ABSOLUTE VAL SMOOTH
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
;*************************************************************************
FRM005                                  COMBINE IMAGES
OVR003   0                              AVERAGE IMAGES
OVR003   1                              SCALED DIFFERENCE
OVR003   2                              ABSOLUTE DIFFERENCE
OVR012   0                              SCALED WGHTD COMBINATION
OVR013   0                              RATIO IMAGES
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
;*************************************************************************
```

```
*****************************************************************
FRM006                                  NOISE REDUCTION
OVR010   0                              MODAL REPLACEMENT
OVR019   1                              ODD DOT
OVR019   0                              ODD LINE
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
*****************************************************************
FRM007                                  EDGE DETECTION
OVR023   0                              POINT EDGE DETECTION
OVR024   0                              AREA EDGE DETECTION
OVR024   1                              AREA EDGE DETECTION MAX
OVR025   0                              SIMPLE FILL IN
OVR025   1                              ADAPTIVE FILL IN
OVR026   0                              CLOSED CURVE
OVR027   0                              LIST BINARY IMAGE
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
*****************************************************************
FRM008                                  TRANSFORMS
OVR029   0                              HADAMARD TRANSF
OVR030   0                              GENERATE FILTER
OVR033   0                              APPLY FILTER
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
*****************************************************************
```

```
;*****************************************************************
FRM009                                  FEATURE EXTRACTION
OVR049   0                              SPECIFY SPECTRAL FILE SET
OVR049   1                              SPECIFY VECTOR FILE SET
OVR048   1                              CREATE SPEC. DESIGN VEC. FILE
OVR048   0                              CREATE SPEC. TEST VEC. FILE
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
;*****************************************************************
FRM010                                  LOGIC DESIGN
OVR045   0                              SELECT VECTOR SET
OVR040   0                              SELECT LOGIC TREE
OVR044   0                              CREATE FISHER LOGIC
OVR044   1                              CREATE BOOLEAN LOGIC
OVR043   0                              EVALUATE LOGIC
OVR050   0                              CREATE THEMATIC MAP
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
;*****************************************************************
FRM011                                  FILE MANIPULATIONS
OVR016   0                              DELETE FILES
OVR017   0                              RENAME FILES
OVR028   1                              SHORT DIRECTORY
OVR028   0                              LONG DIRECTORY
OVR035   0                              TAPE INPUT
OVR035   1                              TAPE OUTPUT
OVR035   2                              SKIP TAPE FILES
OVR035   3                              REWIND TAPE
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
;*****************************************************************
```

```
*************************************************************************
FRM012                                  DISPLAY OPTIONS
OVR014   0                              BINARY DISPLAY
OVR031   0                              DICOMED HDCPY
TRNFRM   0                              MASTER FRAME
END
*************************************************************************
FRM013                                  UTILITY FUNCTIONS
OVR006   0                              DOS
OVR018   0                              LOG
OVR015   0                              CALL USER PROGRAM
OVR001   0                              GENERATE TEST IMAGE
OVR034   0                              CREATE TEST IMAGE
OVR032   0                              EXAMINE HEADER TEXT
OVR027   1                              LIST IMAGE GREY VALUES
TRNFRM   12                             DISPLAY OPTIONS
TRNFRM   0                              MASTER FRAME
END
*************************************************************************
```

APPENDIX D

ERROR MESSAGES

The system error file ERRORS.IPS contains the error messages for all error conditions that might occur during normal system execution. This file is contiguous with a header that has the following format:

Word	Description
Ø	Offset in 32 word blocks to error type Ø
1	Offset in 32 word blocks to error type 1
	⋮

NOTE: Currently, only two error types are defined.

The remainder of the file consists of the error messages, each stored in a 32 word block. The message begins in the first byte of the block. The 37th and 38th characters are filled with a carriage return and line feed respectively. The 64th character contains a null to signal the end of the message.

If the device logical block size is greater than 256 words, then the additional words are not used. This is due to the fact that the memory error buffer is 256 words in length.

The errors are not entered by the user in the above format. Instead the file ERRORS.SRC is edited either via the editor or modification of the card deck. In this file the error entry consists of one line. The messages must be limited to 64 characters. The 37th, 38th, and 64th characters must be blank. The file is in formatted ASCII form and is listed below:

```
E0.0. SPECIFIED FILE IS NONEXISTENT    - RETYPE
E0.1. FILE HAS INCORRECT DATA TYPE     - RETYPE
E0.2. INCORRECT FILE SPECIF. FORMAT    - RETYPE
E0.3. INPUT STRING SYNTAX ERROR        - RETYPE
E0.4. ILLEGAL FRAME NUMBER             - INPUT IGNORED
E0.5. ILLEGAL OPTION NUMBER            - INPUT IGNORED
E0.6. INPUT STRING TOO LONG            - RETYPE
E0.7. PREVIOUS PREMATURE FILE CLOSE    - TYPE CO TO CONTINUE
E0.8. REFERENCED PICTURE ELEMENT       NONEXISTENT - RETYPE
E0.9. DUPLICATE FILE NAME              - RETYPE
E0.10. INCORRECT PARAMETER VALUE       - RETYPE
E0.11. A NONEXISTENT PICTURE ELEMENT   WAS REFERENCED
E0.12. INCORRECT NUMBER OF ENTRIES     - RETYPE
E0.13. INCORRECT FILE FORMAT           - RETYPE
E0.14. ILLEGAL FILE NAME               - RETYPE
E0.15. SPECIFIED FILE IS NONEXISTENT   - INPUT IGNORED
E0.16. ILLEGAL DEVICE                  - RETYPE
E0.17. INCORRECT                       - RETYPE
E0.18. TOO MANY PARENTHESES            - RETYPE
E0.19. ILLEGAL CHAR                    - RETYPE
E0.20. PARENTHESES DO NOT BALANCE      - RETYPE
E0.21. ILLEGAL OPERATOR                - RETYPE
E0.22. OPERAND MISSING                 - RETYPE
E0.23. ARITHMETIC OPERAND EXPECTED     - RETYPE
E0.24. LOGICAL OPERAND EXPECTED        - RETYPE
E0.25. OPERATOR MISSING                - RETYPE
E0.26. ARITH. STATEMENT INCOMPLETE     - RETYPE
E0.27. LOGICAL ARGUMENT INCOMPLETE     - RETYPE
E0.28. ILLEGAL OR INCOMP. STATEMENT    - RETYPE
E0.29. INSUFFICIENT NO. OF POINTS      - INPUT IGNORED
E0.30. FLOATING PT. OVER/UNDER FLOW    - RETYPE
E0.31. ILLEGAL SYMBOL                  - RETYPE
E0.32. ILLEGAL LIMITS                  - RETYPE
E0.33. ILLEGAL MEASUREMENT             - RETYPE
E0.34. DID NOT USE ALL CLASS SYMBOLS   - INPUT IGNORED
```

E1.0. HARDWARE ERROR ON TRANSFER - OPERATION ABORTED
E1.1. CORE SPACE EXHAUSTED - OPERATION ABORTED
E1.2. FILE DIMENSION ERROR - OPERATION ABORTED
E1.3. DUPLICATE FILE NAME - OPERATION ABORTED
E1.4. ILLEGAL DEVICE - OPERATION ABORTED
E1.5. DEVICE FULL - OPERATION ABORTED
E1.6. TEMPORARY FILE ERROR - OPERATION ABORTED
E1.7. SYSTEM TABLE OVERFLOW - OPERATION ABORTED
E1.8. BAD UIC OR DIRECTORY FULL - OPERATION ABORTED
E1.9. DISPLAY COORDINATES ILLEGAL - OPERATION ABORTED
E1.10. NONEXISTENT RECORD REFERENCE - OPERATION ABORTED
E1.11. DOS CORE SPACE EXHAUSTED - OPERATION ABORTED
E1.12. PROTECTION CODE VIOLATION - OPERATION ABORTED
E1.13. FILE OPEN - OPERATION ABORTED
E1.14. LENGTH OR NO. OF RECORDS ZERO - OPERATION ABORTED
E1.15. IMAGE DIMENSION INSUFFICIENT - OPERATION ABORTED
E1.16. ZERO ENTRIES - OPERATION ABORTED
E1.17. ILLEGAL FILE NAME - OPERATION ABORTED
E1.18. A REQUIRED FILE IS NONEXIST. - OPERATION ABORTED
E1.19. INCORRECT FILE FORMAT - OPERATION ABORTED
E1.20. FILE HAS INCORRECT DATA TYPE - OPERATION ABORTED
E1.21. RECORD SEQUENCE ERROR - OPERATION ABORTED
E1.22. LOG FILE CREATE ERROR - OPERATION ABORTED
E1.23. LOG FILE OPEN ERROR - OPERATION ABORTED
E1.24. FAILURE ON "OPEN" OPERATION - OPERATION ABORTED
E1.25. INCORRECT NO. HLP. VARIABLES - OPERATION ABORTED
E1.26. LOG ARG. < OR = TO 0 - OPERATION ABORTED
E1.27. NEIGHBOR OPERATIONS ILLEGAL - OPERATION ABORTED
E1.28. UNEXPECTED END OF FILE - OPERATION ABORTED
E1.29. ILLEGAL LOGIC FILE OPERATION - OPERATION ABORTED
E1.30. CLASS SYMBOLS INCONSISTENT - OPERATION ABORTED
E1.31. NODE DOES NOT EXIST - OPERATION ABORTED
E1.32. NODE NOT LOWEST NODE - OPERATION ABORTED
E1.33. SYMBOLS ALREADY EXIST AT NODE - OPERATION ABORTED
E1.34. ILLEGAL REGION - OPERATION ABORTED
E1.35. MULT. SYMBOLS AT A LOW NODE - OPERATION ABORTED
E1.36. NO CLASS SYMBOLS IN VEC. FILE - OPERATION ABORTED
E1.37. MEASUREMENT IS NONEXISTENT - OPERATION ABORTED

APPENDIX E

DATA FILE DESCRIPTIONS

Several different file formats are used within the system to store various data types. Each type is identified by a different filename extension. The descriptions of each type are found on the following pages.

Certain items in the file headers are common to all types. These items are found in the first and last 64 words of the header. The first 64 words are as follows:

Word	Description
0	Reserved for future use
1	Filename extension (RAD50)
2	Total number of records in file
3	Record length in bytes
4	File close indicator 0 = File was normally closed 1 = File was prematurely closed
5-63	Reserved for future use

The last 64 words of the header are used to store the descriptive text. This text can contain embedded carriage returns and line feeds and is terminated with a null.

File Type: Function File

Extension: .FCT

Special Header Parameters:

Word	Description
64	Set to negative one to show non-array file
65	Function indicator: set to 1 for one-dimensional function. Set to 2 for two-dimensional function.

Data Format:

This file type is used to store the compiled code associated with a filter function. The entire buffer returned by the compiler routine "COMPIL" is stored as one record in this file. Consult "COMPIL" documentation for detailed information. The file format follows (only one record in file):

Word	Description
0	Number of words in this record
1	Offset to compiled code relative to word 0
2	Offset to symbol table relative to word 0
3	Offset to symbol values relative to word 0
4	Filter function in the original ASCII string format
	⋮
	Symbol table
	⋮
	Compiled code
	⋮
	Symbol values

File Type: Filter File

Extension: .FLT

Special Header Parameters:

Word	Description
64.	Array format indicator. Set to 3 to show two-word floating point.

Data Format:

This file type is used to store filter arrays. The elements of the array are in two-word floating point format. One row of elements is stored in one record. The number of records is equal to the number of rows in the filter array.

File Type: Hadamard Transform File

Extension: .HAD

Special Header Parameters:

Word	Description
64.	Array Element Format Indicator 1 = Word elements 2 = Double word elements

Data Format:

This file type is used to store Hadamard transform arrays. The elements of the array are as indicated above. One row of elements is stored in one record. The number of records is equal to the number of rows in the Hadamard array.

File Type: Image File

Extension: .IMG

Special Header Parameters:

Word	Description
64.	Set to zero to indicate integer byte array elements.

Data Format:

This file type is used to store images. One row of picture elements is stored in one record. The number of records is equal to the number of rows in the image.

File Type: Logic Tree

Extension: .LOG

Special Header Parameters: None

Data Format:

This file type is used to store logic trees and their associated logic. Each record in the file is called a page where record 1 is page 0. The records are always 512 bytes in length. The formats of the various page types are detailed on the following pages.

Directory Page - Page 0 (Rec. 1)

Word	Description
0	Number of active entries in directory
1	Number of currently active pages in file
2	Low byte: Logic tree node number if entry is associated with logic. This is set to zero for non-logic entries
Entry 1	High byte: Entry identifier: 1 = One-Space Logic (to be added) 2 = Two-Space Logic (to be added) 3 = Fisher Pairwise Logic 4 = Boolean Logic 5 = Reserved for future logic type. 6 = Logic Tree 7 = Class symbols
3	Page number of block described by this entry
4	Number of pages in this block
	⋮
	Entries continue in the same format as entry 1.

Page 1 (Rec. 2) Reserved for Future Use

Logic Tree Block - Page 2 (Rec. 3)

(Additional Logic Tree Pages are created as required.)

Byte	Description
0	Current page number
1	Number of active entries
2-9	Reserved for future use
10-509	Fifty, ten byte node entries *
510-511	Not used

* Node Entry Format

Byte	Description
0	Node number. Zero if entry is inactive
1	Node level in tree
2-5	Reserved for future use
6	Node number of immediate senior node
7	Number of nodes on next level below this node. Zero indicates that the current node is a terminal node.
8	Node number of first node below
9	Node number of the next node on the current level which has the same immediate senior node. Zero indicates that the current node is the last node on this level.

Note: Node numbers begin at 1 and are assigned consecutively as requested by the logic creation routines.

Node levels also begin at 1 with the senior node and increase consecutively at each lower level.

Class Symbol Block - Page 3 (Rec. 4)

(Additional Class Symbol pages are created as required)

Byte	Description
0	Number of entries on this page
1-511	Class Symbol Entries.* Each entry is of variable length. There is no space between entries.

* Class Symbol Entry Format

Byte	Description
0	Node number in logic tree with which the symbols are associated.
1	Number of symbols in this entry
2	1st class symbol
3	2nd class symbol
	:
	Last class symbol

Fisher Pairwise Logic

A set of Fisher logic corresponding to one node in the logic tree may occupy several consecutive logic file pages. The logic consists of a header block which is stored in one or more pages and the logic components which are also stored in one or more pages. In both cases when extending beyond a page, continuation takes place at the beginning of the next page. The logic components begin at the top of the first page after the header.

Header Format

Word		Description
0,1		Reserved for future use
2		Number of bytes in this header (always even)
3		Offset from top of header to the general information region
4		Offset from top of header to the class symbol region
5		Offset from top of header to the criterion description region
6		Offset from top of header to the measurement description region
7	General Information Region	Minimum vote count
8		Number of the highest dimension with which the logic was created (not necessarily the dimensionality of the vectors)
9,10		Reserved for future use
11	Class Symbol Region	Number of classes
12		Low Byte: Node number to which the class is associated
.		
.		
		High Byte: Class symbol
		.
		.
		.
		(A one word entry appears for each class plus one extra entry for the reject category)

Measurement Description Region	Number of measurements used. Measurement numbers used (one byte per entry) ⋮
	Low Byte: Logic type. This entry allows logic other than Fisher to be used for a given class pair. This is currently set to 3 to indicate Fisher logic
Criterion Description Region	High Byte: Number of thresholds used. The current version only uses the threshold that is half the distance between the class means. Therefore, this is set to 1.
	Page number of logic relative to the first page of the header
	Number of pages in logic
	Relative byte of the start of the logic within the page
	⋮
	(The above four word entry format is repeated for each class pair in the set of classes on which the logic was created.)

Fisher Criterion Format

Word	Description
0	FV_1
.	FV_2
.	.
.	.
	.
	FV_{NDIM}
	Fisher vector in two-word floating point format. NDIM is the largest dimension with which the logic was created. This is not necessarily the dimensionality of the vectors.
	DV_1
	DV_2
	.
	.
	.
	DV_{NDIM}
	This is the discriminant vector for the above Fisher vector. This is not currently used, but is stored for future use.
	θ_1
	θ_2
	θ_3
	θ_4
	θ_5
	Two-word floating point threshold values.*
	.
	.
	.
	(The above is repeated for each class pair.)

* Five thresholds are computed and stored for each class pair. Only threshold three is currently used by logic evaluation. The other thresholds are stored for possible future software additions. The thresholds are computed as follows:

Let μ_i be the estimated mean of class i projected onto the Fisher direction $\underset{\sim}{d}$, i.e.,

$$\mu_i = \underset{\sim}{d}^T \overline{\underset{\sim}{X}}_i \qquad i = 1,2$$

where $\overline{\underset{\sim}{X}}_i$ is the mean of class i.

Then the thresholds are:

$$\theta_1 = \mu_2 - \Delta/2$$

$$\theta_2 = \mu_2 + \Delta/3$$

$$\theta_3 = \mu_2 + \Delta/2$$

$$\theta_4 = \mu_1 - \Delta/3$$

$$\theta_5 = \mu_1 + \Delta/2$$

where

$$\Delta = \mu_1 - \mu_2$$

Boolean Logic Format

Word	Description
0,1	Reserved for future use
2	Number of bytes in logic
3	Low Byte: Node number of false node High Byte: Node number of true node
4	Address of compiled code relative to word 0
5	Address of symbol table relative to word 0
6	Address of symbol values relative to word 0
7	Input ASCII character string terminated with a null
	.
	.
	.
	Symbol Table
	.
	.
	.
	Compiled Code
	.
	.
	.
	Symbol Values

Symbol Table Format

Word	Definition
0	Number of symbols
2	Status word for symbol 1
3,4	RAD50 pack of symbol 1
5	Status word for symbol 2
6,7	RAD50 pack of symbol 2
	.
	.
	.
	Status word for last symbol
	RAD50 pack of last symbol

Status Word Format

Low Byte:	This is always set to 1. Other values returned by the compiler are not accepted by the logic creation routine. (See COMPIL for other values.)
High Byte:	Set to the measurement number (binary) which the symbol represents.

Calling the Compiled Code

The compiled code is called as follows:

```
JSR       PC,(adr. of code)
```

Prior to this call, the values of all measurements for the current vector must be provided. These values are placed in the buffer area identified by word 6 of the Boolean logic. The first four words of this area are reserved for special subroutine addresses. The first three should be filled with the entry point addresses of the EXP, LN and LOG routines respectively. The fourth is a special routine for image neighborhood operations. This feature of the compiler will not occur in the logic. Therefore, this word is not used.

Following the four special addresses appear the measurement values in the order of appearance within the symbol table. All values are in two-word floating point. The logical result is returned in R1 as 1 for true or a 0 for false.

AD-A038 137 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
IMAGE PROCESSING SOFTWARE CONVERSION. COMPUTER PROGRAM DOCUMENT--ETC(U)
FEB 77 J C LIETZ, M J GILLOTTE, D R HOUSE F30602-76-C-0164
UNCLASSIFIED PAR-76-35-VOL-2 RADC-TR-77-51-VOL-2 NL

3 OF 3
AD A038137

END

DATE FILMED
4-77

File Type: Mean-Covariance File

Extension: .MC

Special Header Parameters: None

Data Format:

This file type is used to store the mean vectors and covariance matrices from which Fisher logic is created. One record exists in the file for each class. The record format follows:

Word	Description
0	Class symbol in lower byte. Upper byte unused.
1,2	Two-word floating point count of the number of vectors used in computing the mean and covariance
3	MV_1, MV_2, . . . MV_{NDIM} — Two-word floating point mean vector. NDIM is the number of dimensions in the vector set.
	Upper Triangle of covariance matrix This is stored by row beginning with the diagonal element.

File Type: Region File

Extension: .REG

Special Header Parameter: None

Data Format:

This file type is used to store region boundary description. The coordinate pairs of the vertices are stored one each in a four byte record. Each record contains the row coordinate value in the first two bytes and the column coordinate value in the last two bytes. The last record contains the same coordinate pair as the first record. This forces a closed boundary.

File Type: Spectral Set File

Extension: .SPC

Special Header Parameters:

Word	Description
64	Always 0
65	Number of image file names
66	Number of region file names

Data Format:

This file type is used to store the description of a set of spectral images. Also included are names of region files from which vectors can be created. The record length is 14 bytes where each record contains an image file specification or one region file specification. Image file specifications appear first, followed by the region file specifications. The formats are as follows:

Spectral Set File Record Format

Image File Names:

Word	Description
0	Device name (RAD50)
1	Unit No.
2-3	File Name (RAD50)
4	Extension (RAD50)
5	User ID
6	Not Used
7	Not Used

Region File Names:

Word	Description
0	Device Name (RAD50)
1	Unit No.
2-3	File Name (RAD50)
4	Extension (RAD50)
5	User ID
6	Class Symbol
7	Data Reduction Factor

File Type: Vector Set File

Extension: .VCS

Special Header Parameters: None

Data Format:

This file type is used to store the description of a set of vector files. The record length is 14 bytes where each record contains one vector file specification as follows:

Word	Description
0	Device Name (RAD50)
1	Unit No.
2-3	File Name (RAD50)
4	Extension (RAD50)
5	User ID
6	Not Used
7	Not Used

File Type: Vector File

Extension: .VEC

Special Header Parameters:

Word	Description
64	Meas. Format Ind. 0 = Integer Byte 1 = Integer Word 2 = (Not used) 3 = Flt. Pt. (Single precision) 4 = (Not used)
65	Vector Header Format Indicator Bit 0: 0 = Perm. and Temp. Sym. Space not Allocated 1 = Perm and Temp. Sym. Space Allocated Bit 1: 0 = Row and Column not present 1 = Row and Column present
66	Total number of meas. per vector
67	Number of vectors per record
68-69	Total number of vectors
70	Row dimension of original image
71	Column dimension of original image
72	Vector length in bytes (always even)
73	Number of spectral measurements

Data Format:

This file type is used to store measurement vectors. Each vector file corresponds to one image file from which the vectors were extracted. Depending upon the total number of vectors, one or more vectors may be stored in a record. This is done to keep the record count to single precision. When more than one vector is stored per record, the records are packed with no intervening space. The format of the vector is as follows:

Word	Description
0 . . .	M_1 M_2 . . . M_N Measurements (see header word 64 for format) Low Byte: Node number High Byte: Not used

Low Byte: Temporary class symbol
High Byte: Permanent class symbol
Row number in image at which vector was found
Column number in image at which vector was found

APPENDIX F

SUBROUTINE LIBRARY

The following is a listing of the subroutine library which is stored in file "LIBR.OBJ":

```
LIBR  V05A

      LIBR  .LS        03-DEC-76  11:08:00

      SEQ.     NAME   VERSION

      00001    HADMAR  V001
      00002    SMOOTH  V001
      00003    TRNFTN  V001
      00004    HISPLT  V001
      00005    COMPIL  V001
      00006    EVALFP  V001
      00007    EVALBL  V001
      00008    SPAMEZ  V001
      00009    PARCOR  V001
      00010    FINDPT  V001
      00011    LOGGER  V001
      00012    VECFNS  V001
      00013    CRLOG   V001
      00014    SQRT    V001
      00015    OPTION  V001
      00016    CB2DA   V001
      00017    RAD2AS  V001
      00018    AS2RAD  V001
      00019    LNPRNT  V001
      00020    LOG     V001
      00021    EXP     V001
      00022    FPASC   V001
      00023    CHCUR   V001
      00024    TMPARA  V001
      00025    FPSQRT  V001
      00026    FRSUB
```

The following is the batch stream that is used to generate library LIBR.OBJ. This batch stream is stored in file "LIBBLD.BAT".

```
$JOB LIBBLD[11,11],KB:
$R PIP
*L:=BP.OBJ,L1.BLD,L2.BLD,L3.BLD,L4.BLD,L5.BLD/DE
*L1.BLD<HADMAR.OBJ,SMOOTH.OBJ,TRNFTN.OBJ,HISPLT.OBJ,COMPIL.OBJ
*L2.BLD<EVALFP.OBJ,EVALBL.OBJ,SFAMEZ.OBJ,FAKCOR.OBJ,FINDFT.OBJ
*L3.BLD<LOGGER.OBJ,VECFNS.OBJ,CRLOG.OBJ,SQRT.OBJ,OPTION.OBJ
*L4.BLD<CR2DA.OBJ,RAD2AS.OBJ,AS2RAD.OBJ,LNPRNT.OBJ,LOG.OBJ,EXP.OBJ
*L5.BLD<FPASC.OBJ,CHOOR.OBJ,TMPARA.OBJ,FPSORT.OBJ,FPSUB.OBJ
$R LIBR
*LIBR.OBJ<*,L1.BLD,L2.BLD,L3.BLD,L4.BLD,L5.BLD
*C PIP
*L1.BLD,L2.BLD,L3.BLD,L4.BLD,L5.BLD/DE
*FINISH
```

APPENDIX G

SYSTEM BUILDING BATCH STREAMS

This appendix contains listings of the batch streams and the overlay descriptor files. These are all used to build the image processing system. Two versions of the batch streams are available. The first builds the system without the ODT routine. This version of the system is used for normal user appplications. The second batch stream includes ODT at the expense of a smaller buffer area. This version should be used only for debugging because it executes more slowly. Both batch streams call the common batch stream "IPSBLD.BAT" to link the option overlays.

The following batch stream, which is stored in "IPS.BAT", builds the system without ODT:

```
$JOB IPS[11,11],KB:
$R PIP
#IPS.LDA,IPS.MAP,IPS=STB/DE
$R LINK
#IPS,IPS,DK:IPS.STB<IPS.ODL/MP/B:35700/0
#EXTSCT=FRECOR:104000              ;SET FRECOR AT 17K
#GBLDEF=CSIZE:42000                ;SIZE OF FRECOR IN WDS.
#GBLDEF=CSIZE2:104000              ;SIZE OF FRECOR IN BYTES
#GBLDEF=DEFDV:15270                ;DEFAULT DEV. = DK:
#GBLDEF=DEFUN:1                    ;DEFAULT UNIT = 1
#/E
$CH IPSBLD.BAT[11,11]
$FINISH
```

The following batch stream, which is stored in file "IPSODT.BAT", builds the system with ODT:

```
#$JOB IPSODT[11,11],KB:
$R PIP
#IPS.LDA,IPS.MAP,IPS,STB/DE
$R LINK
#IPS,IPS,DK:IPS.STB<IPSODT.ODL/MP/B:35700/0
#EXTSCT=FRECOR:77000               ;SET FRECOR AT 16K - 256.
#GBLDEF=CSIZE:37400                ;SIZE OF FRECOR IN WDS.
#GBLDEF=CSIZE2:77000               ;SIZE OF FRECOR IN BYTES
#GBLDEF=DEFDV:15270                ;DEFAULT DEV. = DK:
#GBLDEF=DEFUN:1                    ;DEFAULT UNIT = 1
#/E
$CH IPSBLD.BAT[11,11]
$FINISH
```

The following "change" batch stream, which is stored in file "IPSBLD.BAT", is called by both of the above batch streams:

```
$R FTP
#OVR001,LDA+OVR002,LDA+OVR003,LDA+OVR004,LDA+OVR005,LDA/DE
#OVR006,LDA+OVR007,LDA+OVR008,LDA+OVR009,LDA+OVR010,LDA/DE
#OVR011,LDA+OVR012,LDA+OVR013,LDA+OVR014,LDA+OVR015,LDA/DE
#OVR016,LDA+OVR017,LDA+OVR018,LDA+OVR019,LDA+OVR020,LDA/DE
#OVR021,LDA+OVR022,LDA+OVR023,LDA+OVR024,LDA+OVR025,LDA/DE
#OVR026,LDA+OVR027,LDA+OVR028,LDA+OVR029,LDA+OVR030,LDA/DE
#OVR031,LDA+OVR032,LDA+OVR033,LDA+OVR034,LDA+OVR035,LDA/DE
#OVR036,LDA+OVR037,LDA+OVR038,LDA+OVR039,LDA+OVR040,LDA/DE
#OVR041,LDA+OVR042,LDA+OVR043,LDA+OVR044,LDA+OVR045,LDA/DE
#OVR046,LDA+OVR047,LDA+OVR048,LDA+OVR049,LDA+OVR050,LDA/DE
FIN
```

```
#OVR001/CO<IPS.STB+GTFILE+LIBR/T:35700/E
#OVR002/CO<IPS.STB+ARSLCT+LIBR/T:35700/E
#OVR003/CO<IPS.STB+COMBIM+LIBR/T:35700/E
#OVR004/CO<IPS.STB+SPBIAS+LIBR/T:35700/E
#OVR005/CO<OVR005.ODL/MP/T:35700/E
#OVR006/CO<IPS.STB+UTILTY/T:35700/E
#OVR007/CO<IPS.STB+ELCHNG+LIBR/T:35700/E
#OVR008/CO<IPS.STB+NORMLZ+LIBR/T:35700/E
#OVR009/CO<IPS.STB+SHIFTI+LIBR/T:35700/E
#OVR010/CO<IPS.STB+MODREP+LIBR/T:35700/E
#OVR011/CO<IPS.STB+DECIM+LIBR/T:35700/E
#OVR012/CO<IPS.STB+DIFFAC+LIBR/T:35700/E
#OVR013/CO<IPS.STB+RTDOMG+LIBR/T:35700/E
#OVR014/CO<IPS.STB+TEKDSP+LIBR/T:35700/E
#OVR015/CO<IPS.STB+CALUSP/T:35700/E
#OVR016/CO<IPS.STB+DELETE/T:35700/E
#OVR017/CO<IPS.STB+RENAME/T:35700/E
#OVR018/CO<IPS.STB+LOGOVR/T:35700/E
#OVR019/CO<IPS.STB+ODDLD+LIBR/T:35700/E
#OVR020/CO<IPS.STB+PWLINR+LIBR/T:35700/E
#OVR021/CO<IPS.STB+SIMSMO+LIBR/T:35700/E
#OVR022/CO<IPS.STB+WTDSMO+LIBR/T:35700/E
#OVR023/CO<IPS.STB+FTEDGE+LIBR/T:35700/E
#OVR024/CO<IPS.STB+LINES+LIBR/T:35700/E
#OVR025/CO<IPS.STB+FILL+LIBR/T:35700/E
#OVR026/CO<IPS.STB+CLCV+LIBR/T:35700/E
#OVR027/CO<IPS.STB+LPHDCP+LIBR/T:35700/E
#OVR028/CO<IPS.STB+DIRCTY+LIBR/T:35700/E
#OVR029/CO<IPS.STB+HADXFM+LIBR/T:35700/E
#OVR030/CO<OVR030.ODL/MP/T:35700/E
#OVR031/CO<IPS.STB+DTCOMD+LIBR/T:35700/E
#OVR032/CO<IPS.STB+CKTEXT+LIBR/T:35700/E
#OVR033/CO<IPS.STB+FLTMUL+LIBR/T:35700/E
#OVR034/CO<IPS.STB+CTFILE+LIBR/T:35700/E
#OVR035/CO<IPS.STB+TAPE/T:35700/E
```

```
#OVR036/CO<IPS.STB,KEYARA,LIBR/T:35700/E
#OVR040/CO<IPS.STB,SELOGF,LIBR/T:35700/E
#OVR041/CO<IPS.STB,MENCOV,LIBR/T:35700/E
#OVR042/CO<OVR042.ODL/MF/T:35700/E
#OVR043/CO<IPS.STB,EVALDR,LIBR/T:35700/E
#OVR044/CO<IPS.STB,SELNOD/T:35700/E
#OVR045/CO<IPS.STB,SELVEC,LIBR/T:35700/E
#OVR046/CO<IPS.STB,GETBOL,LIBR/T:35700/E
#OVR047/CO<IPS.STB,RGSTAT,LIBR/T:35700/E
#OVR048/CO<IPS.STB,CREVEC,LIBR/T:35700/E
#OVR049/CO<IPS.STB,CRESFS,LIBR/T:35700/E
#OVR050/CO<IPS.STB,DECIMG,LIBR/T:35700/E
```

The following is the overlay descriptor file "IPS.ODL" which is used in building the system without ODT:

```
OVR01:  .FCTR   FILE1
OVR02:  .FCTR   FILE2
OVR03:  .FCTR   FILE3
OVR04:  .FCTR   FILE4
MAIN:   .FCTR   PLOT-TTYIO-TELEIO-SAVER-SARFPS-BLDISP-EXEC
        .ROOT   MAIN-FTNLIB-*(OVR01,OVR02,OVR03,OVR04)
        .END
```

The following is the overlay descriptor file "IPSODT.ODL" which is used in building the system with ODT:

```
OVR01:  .FCTR   FILE1
OVR02:  .FCTR   FILE2
OVR03:  .FCTR   FILE3
OVR04:  .FCTR   FILE4
MAIN:   .FCTR   PLOT-TTYIO-TELEIO-SAVER-SARFPS-BLDISP-EXEC
        .ROOT   MAIN-FTNLIB-ODT/OD-*(OVR01,OVR02,OVR03,OVR04)
        .END
```

The following is the overlay descriptor file "OVRØØ5.ODL": which is used in building overlay "OVRØØ5.LDA":

```
OVR01:  .FCTR   FINDPT-OPTION
OVR02:  .FCTR   HISPLT-SQRT-CB2DA-RAD2AS-FPASC
        .ROOT   IPS.STB-DENHST-*(OVR01,OVR02)
        .END
```

The following is the overlay descriptor file "OVRØ3Ø.ODL" which is used in building overlay "OVRØ3Ø.ODL".

```
.FCTR   COMPIL-LINK
.FCTR   METHOD-FPASC-FPSQRT-EXP-LOG-CB2DA-RAD2AS
.FCTR   IPS.STB-LREFLT
.ROOT   MAIN-$(OVR01,OVR02)
.END
```

The following is the overlay descriptor file "OVRØ42.ODL" which is used in building overlay "OVRØ42.ODL".

```
.FCTR   GETMEA
.FCTR   INVMAT
.F ??   FPM-FPSCMP-LOGARI-GETFIX-INVERT-CMPRES-MULTV-FPSQRT
.ROOT   IPS.STB-MAIN-$(OVR01,OVR02)
.END
```

APPENDIX H

SYSTEM MACROS

This appendix contains a listing of the system macros which are stored in file "MACS.MAC". When a program references any of these macros, this file must be specified as the first input file in the assembler command line. The following is an example of its use:

```
DK:,LP: < MACS,PROG
```

where the file "PROG" is a program that references one or more of the system macros.

The listing of MACS.MAC is as follows:

```
        .NLIST
;    THIS FILE CONTAINS ALL THE GENERAL MACRO DEFINITIONS FOR THE
;IMAGE PROCESSING SYSTEM.
;    LISTING OF THE MACRO DEFINITIONS IS SUPPRESSED THROUGHOUT
;THIS FILE.
;
;*****************************************************************
;THIS MACRO ESTABLISHES THE SYSTEM PARAMETERS
        .MACRO  SYSTEM
;
;THE FOLLOWING DEFINES THE FIRST THREE CHARS. OF AN OVERLAY NAME
        OVRNAM=56700+1560+22    ;O+V+R (RAD50)
;THE FOLLOWING DETERMINES THE DEVICE ON WHICH IPS RESIDES
        SYDEV=73300+1750        ;S+Y (RAD50)
;THE FOLLOWING DETERMINES THE UNIT ON WHICH IPS RESIDES
        UNITNO=0                ;UNIT 0
;THE FOLLOWING DETERMINES THE USER NUMBER UNDER WHICH OVERLAYS
;ARE SEARCHED.
        IPSUID=4411             ;UID=11,11
        .ENDM   SYSTEM
```

```
;*****************************************************************
;    THE MACRO F.REQ GENERATES THE FILE REQUEST BLOCK FOR ALL
;FILE FUNCTIONS.  THE POSSIBLE ARGUMENTS ARE:
;      LBL=LABEL PREFIX
;      CHR=ADR OF CHAR. STRING (0=NAME SUPPLIED IN REQUEST BLK)
;      EXT=FILE EXTENSION (DATA TYPE)
;      RECS=TOTAL NO. OF RECORDS
;      LNG=RECORD LENGTH (BYTES)
;      ACES=ACCESS TYPE
;         1-READ
;         2-WRITE
;         3-MODIFY (R/W)
;      FTYP=FILE TYPE
;         0-CONTIGUOUS
;         1-LINKED
;      RTYP=RECORD TYPE
;         0-FIXED LENGTH
;         1-VARIABLE LENGTH
;      NAME=FILE NAME.  THIS ARG. MUST BE LEFT BLANK IF NO NAME
;           IS GIVEN.
;
;THE CALL FORMAT FOR THE CREATE FUNCTION IS:
;        F.REQ   0,LBL,CHR,EXT,RECS,LNG,ACES,FTYP,RTYP,NAME
;
;THE CALL FORMAT FOR THE RETRIEVE FUNCTION IS:
;        F.REQ   1,LBL,CHR,EXT,ACES,NAME
;
;
          .MACRO  F.REQ A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
          .IF     EQ A1                       ;CREATE
         F.REQ1   A2,A3,A4,A5,A6,A7,A8,A9,A10
          .MEXIT
          .ENDC
          .IF     EQ A1-1                     ;RETRIEVE
         F.REQ1   A2,A3,A4,0,0,A5,0,0,A6
          .MEXIT
          .ENDC
          .ENDM   F.REQ
```

```
;*********************************************************************
;THE MACRO F.REQ1 IS CALLED BY F.REQ TO SET UP THE REQUEST BLOCK
        .MACRO  F.REQ1 A,B,C,D,E,F,G,H,I
A'CHR:  .WORD   B                       ;ADR OF OUTPUT CHAR STRING
        .IF     NB,I
        .IFT
A'NAM:  .RAD50 /I/                      ;FILE NAME
        .NCHR   F.CHRS,<I>
        .IIF    LE,F.CHRS-3,            .WORD    0
        .IFF
A'NAM:  .WORD   0,0                     ;FILE NAME
        .ENDC
A'EXT:  .RAD50  /C/                     ;EXTENSION TO SHOW DATA T
A'DEV:  .WORD   0                       ;DEVICE IN RAD50
A'UNT:  .BYTE   0                       ;UNIT NO.
A'RTF:  .BYTE   <4*F>+<2*G>+H           ;RECORD, FILE AND ACCESS
                                        ;   BIT 0=0 - FIXED LENGTH REC.
                                        ;         =1 - VAR. LENGTH REC
                                        ;   BIT 1=0 - CONTIG. FILE
                                        ;         =1 - LINKED FILE
                                        ;   BIT 2=0 - NO READ
                                        ;         =1 - READ
                                        ;   BIT 3=0 - NO WRITE
                                        ;         =1 - WRITE
A'BUF:  .WORD   FRECOR                  ;BUFFER ADR.
A'SIZ:  .WORD   CSIZE2                  ;BUFFER SIZE (BYTES)
A'CNT:  .WORD   D                       ;TOTAL NO. OF RECORDS
A'LEN:  .WORD   E                       ;RECORD LENGTH
A'CON:  .WORD   1                       ;NO. OF CONTIG. REC. REQUIRED
A'REC:  .WORD   0                       ;REC. NO. REQUESTED
        .ENDM   F.REQ1
;*********************************************************************
;   THE F.CRE GENERATES A SUBROUTINE CALL TO CREATE AND OPEN A
;FILE FOR I/O.  THE HEADER WILL BE ESTABLISHED IN CORE BY F.CRE.
;ITS ADDRESS WILL BE RETURNED IN R0.
;     ADR=ADR OF REQUEST BLOCK (1ST WD)
;     ALTRET=ALTERNATE RETURN ADR. WHEN CR TYPED (OPTIONAL)
;
        .MACRO  F.CRE ADR,ALTRET
        JSR     R5,F.CRE
        .NARG   X$$
        BR      .+<2*X$$>+2
        .NLIST  CND
        .IIF NB ADR,            .WORD   ADR
        .IIF NB ALTRET,             .WORD   ALTRET
        .LIST   CND
        .ENDM   F.CRE
```

```
;*************************************************************************
;    THE MACRO F.CRE$ GENERATES A SUBROUTINE CALL TO CREATE A FILE.
;THE HEADER WILL BE ESTABLISHED IN CORE AND ITS ADDRESS WILL BE RETURNED
;R0
;       ADR=ADR. OF REQUEST BLOCK
;       ALTRET=ALTERNATE RETURN ADR WHEN CR TYPED
;
          .MACRO   F.CRE$ ADR,ALTRET
          JSR      R5,F.CRE$
          .NARG    X$$
          BR       .+<2*X$$>+2
          .NLIST   CND
          .IIF NB ADR,          .WORD    ADR
          .IIF NB ALTRET,          .WORD    ALTRET
          .LIST    CND
          .ENDM    F.CRE$
;*************************************************************************
;    THE MACRO F.RET GENERATES A SUBROUTINE CALL TO RETRIEVE AND
;OPEN A FILE FOR I/O.  THE HEADER WILL BE READ INTO CORE BY F.RET.
;ITS ADDRESS WILL BE RETURNED IN R0.
;       ADR=ADR.OF REQUEST BLOCK (1ST WORD)
;       ALTRET=ALTERNATE ADR. WHEN CR TYPED
;
           .MACRO   F.RET ADR,ALTRET
          JSR      R5,F.RET
          .NARG    X$$
          BR       .+<2*X$$>+2
          .NLIST   CND
          .IIF NB ADR,          .WORD    ADR
          .IIF NB ALTRET,          .WORD    ALTRET
          .LIST    CND
           .ENDM    F.RET
;*************************************************************************
;    THE MACRO F.RET$ GENERATES A SUBROUTINE CALL TO RETRIEVE A FILE.
;THE HEADER WILL BE ESTABLISHED IN CORE AND ITS ADDRESS WILL BE RETURNED
;IN R0.
;       ADR=ADR. OF REQUEST BLOCK
;       ALTRET=ALTERNATE RETURN ADR WHEN CR TYPED
;
          .MACRO   F.RET$ ADR,ALTRET
          JSR      R5,F.RET$
          .NARG    X$$
          BR       .+<2*X$$>+2
          .NLIST   CND
          .IIF NB ADR,          .WORD    ADR
          .IIF NB ALTRET,          .WORD    ALTRET
          .LIST    CND
          .ENDM    F.RET$
```

```
;*****************************************************************
;THE MACRO F.CHK SEARCHES FOR A FILE TO DETERMINE WHETHER OR NOT IT
;EXISTS.
;       ADR=ADR. OF REQUEST BLOCK
;       NOFILE=RETURN ADDRESS IF FILE NONEXISTENT
;
          .MACRO   F.CHK ADR,NOFILE
          JSR      R5,F.CHK
          BR       .+6
          .WORD    ADR                ;REQUEST BLK. ADR.
          .WORD    NOFILE             ;RETURN ADR. IF FILE NONEXISTENT
          .ENDM    F.CHK
;*****************************************************************
;    THE MACRO F.DEL GENERATES A SUBROUTINE CALL TO DELETE A
;FILE.
;       ADR=ADR OF BUFFER WITH LINK AND FILE BLKS
;       NOFILE=RETURN ADDRESS IF FILE NONEXISTENT.  FATAL IF 0.
;
           .MACRO   F.DEL ADR,NOFILE
           CALL     F.DEL,ADR, NOFILE
           .ENDM    F.DEL
;*****************************************************************
;    THE MACRO F.RNM GENERATES A SUBROUTINE CALL TO RENAME A FILE.
;       ADR=ADR OF BUFFER WITH LINK AND FILE BLKS
;       NOFILE=RETURN ADDRESS IF FILE NONEXISTENT.  FATAL IF 0.
;       DUPFIL=RETURN ADDRESS IF DUP. FILE NAME.  FATAL IF 0.
;
           .MACRO   F.RNM ADR,NOFILE,DUPFIL
           CALL     F.RNM,ADR,NOFILE,DUPFIL
           .ENDM    F.RNM
;*****************************************************************
;    THE MACRO F.PTR GENERATES A SUBROUTINE CALL TO RETRIEVE THE CORE
;ADR. OF A GIVEN RECORD OR RECORDS.  THE ADDRESSES ARE RETURNED
;ON THE STACK.  THE FIRST REC. ADR. IS ON THE TOP FOLLOWED BY
;THE REMAINING REC. ADRS. IF ANY.  NOTHING IS RETURNED IF THE
;RECORD IS NONEXISTENT.
;       ADR=ADR OF REQUEST BLK (1ST WD)
;       ERROR=RECORD NONEXISTENT
;
           .MACRO   F.PTR ADR,ERROR
          .NARG     .SYM
          .IF       EQ,.SYM-2
           CALL      F.PTR,ADR,ERROR
          .MEXIT
          .ENDC
          .ERROR    .SYM              ;ILLEGAL NO. OF ARGUMENTS
           .ENDM    F.PTR
```

```
;*****************************************************************
;    THE MACRO F.CLOS GENERATES A SUBROUTINE CALL TO CLOSE A FILE
;      ADR=ADR OF REQUEST BLK (1ST WD)
;
          .MACRO   F.CLOS ADR
          JSR      R5,F.CLOS                ;CLOSE A FILE
          BR       .+4
          .WORD    ADR                      ;REQ. BLK ADR.
          .ENDM    F.CLOS
;*****************************************************************
;    THE MACRO F.SHUT GENERATES A SUBROUTINE CALL TO CLOSE ALL FILES
;CURRENTLY OPEN.
;
          .MACRO   F.SHUT
          CALL     F.SHUT                   ;CLOSE ALL OPEN FILES
          .ENDM    F.SHUT
;*****************************************************************
;THIS MACRO EXTENDS A FILE BY A SPECIFIED NO. OF BLOCKS
;        ADR=ADR. OF REQUEST BLOCK
;        EXTEND=ADR. OF EXTENSION IN 256 WD. BLKS.
;
          .MACRO   F.EXT,ADR,EXTEND
          JSR      R5,F.EXT
          BR       .+6
          .WORD    ADR                ;REQUEST BLK. ADR.
          .WORD    EXTEND             ;EXTENSION
          .ENDM    F.EXT
;*****************************************************************
;    THE MACRO CALL GENERATES AN EXTERNAL SUBROUTINE CALL WITH UP TO
;SIX PARAMETERS.
;
          .MACRO   CALL NAM,P1,P2,P3,P4,P5,P6
          .NARG    X$$
          JSR      R5,NAM
          .IF      NB P1
          BR       .+X$$+X$$
          .ENDC
          .NLIST   CND
          .IIF NB P1,          .WORD   P1
          .IIF NB P2,          .WORD   P2
          .IIF NB P3,          .WORD   P3
          .IIF NB,P4,          .WORD   P4
          .IIF NB P5,          .WORD   P5
          .IIF NB P6,          .WORD   P6
          .LIST    CND
          .ENDM    CALL
```

```
;*****************************************************************
;    THE MACRO ER.FAT GENERATES A CALL TO THE ERROR REPORTING
;ROUTINE FOR FATAL ERRORS.
;       NUM=FATAL ERROR NUMBER
;
          .MACRO   ER.FAT NUM
          JSR      R5,E.ERR
          BR       .+4
          .BYTE    NUM                          ;FATAL ERROR NUMBER
          .BYTE    1                            ;FATAL ERROR
          .ENDM    ER.FAT
;*****************************************************************
;    THE MACRO ER.REC GENERATES A CALL TO THE ERROR REPORTING ROUTINE
;FOR RECOVERABLE ERRORS
;       NUM=RECOVERABLE ERROR NUMBER
;
          .MACRO   ER.REC NUM
          JSR      R5,E.ERR
          BR       .+4
          .BYTE    NUM                          ;RECOVERABLE ERROR NUMBER
          .BYTE    0                            ;RECOVERABLE ERROR
          .ENDM    ER.REC
;*****************************************************************
;    THE MACRO MACS DEFINES THE .MCALL'S FOR ALL MACROS IN THE
;DUS SYSMAC.SML FILE.
;
          .MACRO   MACS
          .MCALL   .PARAM,.INIT,.RLSE,.CLOSE,.READ,.WRITE
          .MCALL   .OPENO,.OPENI,.OPENU,.OPENC,.OPENE,.WAIT,.BLOCK
          .MCALL   .WAITR,.TRAN,.SPEC,.STAT,.ALLOC,.DELET,.RENAM
          .MCALL   .APPND,.LOOK,.KEEP,.EXIT,.TRAP,.STFPU,.RECRD
          .MCALL   .DUMP,.RSTRT,.CORE,.MONR,.MONF,.DATE,.TIME
          .MCALL   .GTUIC,.SYSDV,.RADPK,.RADUP,.D2BIN,.BIN2D,.O2BIN
          .MCALL   .BIN2O,.CSI1,.CSI2,.DTCVT,.TMCVT,.CVTDT,.STPLA
          .MCALL   .GTPLA,.DTCIV,.GTSTK,.STSTK,.RUN,.FLUSH,.OPEN
         .MCALL   .GTRDV,.GTOVF
          .ENDM    MACS
;*****************************************************************
;    THIS MACRO DEFINES THE FLOATING POINT REGISTER MNEMONICS.
         .MACRO   REGS
         AC0=%0
         AC1=%1
         AC2=%2
         AC3=%3
         AC4=%4
         AC5=%5
         .ENDM
```

```
;*************************************************************************
;MACRO FOR DEFINING LINK BLOCK
;
        .MACRO  LNKBLK          CHR,RET,LDN,NWTF,UN,PDN
; LINK BLOCK DEFINITION
        .NLIST  CND
        .WORD   RET             ;ERROR RETURN ADDRESS,0=FATAL
CHR'LNB: .WORD   0               ;LINK POINTER
.IIF B,LDN,         .RAD50  / /             ;NO LOGICAL NAME
.IIF NB,LDN,         .RAD50  /LDN/           ;LOGICAL DATASET NAME
        .BYTE   NWTF            ;NUMBER OF WDS TO FOLLOW
        .BYTE   UN              ;UNIT NUMBER
        .IF     EQ,NWTF-1
.IIF B PDN,         .RAD50  / /             ;ASSIGN MUST BE USED
.IIF NB PDN,         .RAD50  /PDN/           ;PHYSICAL DATASET NAME
        .ENDC
        .LIST CND
        .ENDM
;*************************************************************************
;   THIS MACRO DEFINES FILE NAME BLOCKS.
        .MACRO  FILBLK          CHR,RET,HO,FILNAM,EXT,UIC,PC
;FILENAME BLOCK DEFINITION
        .NLIST  CND
        .WORD   RET             ;ERROR RET. ADDRESS, 0=FATAL
        .BYTE   HO              ;HOW OPEN CODE
CHR'ERC: .BYTE   0               ;ERROR CODE
        .IF NB,FILNAM
        .IFF
CHR'FNB: .WORD   0,0,0           ;FILE NAME,EXT NOT REQUIRED
        .IFT
CHR'FNB: .RAD50  /FILNAM/        ;FILE NAME
        .RAD50  /EXT/
        .ENDC
        .WORD   UIC             ;USER ID CODE
        .WORD   PC              ;PROTECTION CODE
        .LIST   CND
        .ENDM
;*************************************************************************
;   THIS MACRO DEFINES BUFFER HEADERS.
        .MACRO  BUFHDR CHR,MBCNT,MODE,ABCNT,PNTR
;LINE BUFFER HEADER DEFINITION
CHR'MBC: .WORD   MBCNT           ;MAXIMUM BYTE COUNT
        .BYTE   MODE            ;TRANSFER MODE
        .BYTE   0               ;TRANSFER STATUS BYE
CHR'ABC: .WORD   ABCNT           ;ACTUAL BYTE COUNT
        .NLIST  CND
        .IF NB,PNTR
CHR'PTR: .WORD   PNTR            ;STARTING ADR OF LINE BUFFER
        .ENDC
        .LIST   CND
        .ENDM
```

```
;*************************************************************************
;    THIS MACRO DEFINES TRAN BLOCKS.
        .MACRO  TRNBLK  CHR,DBN,BUFR,WCNT,FUNC
;TRAN BLOCK DEFINITION
CHR'TNB: .WORD   DBN                ;DEVICE BLK NO.
CHR'BUF: .WORD   BUFR               ;BUFFER ADR
CHR'CNT: .WORD   WCNT               ;WORD COUNT TO TRANSF.
        .BYTE   FUNC               ;FUNCTION CODE BYTE
CHR'TST: .BYTE   0                  ;STATUS BYTE
        .WORD   0                  ;NUMBER OF WDS NOT TRANSFERRED
        .ENDM
;*************************************************************************
;    THIS MACRO DEFINES RECORD BLOCKS.
        .MACRO  RECBLK  CHR,FUNC,BUF,RLEN,HORN,LORN
; RECORD BLOCK DEFINITION
CHR'RCB:        .BYTE  FUNC        ;FUNCTION CODE
        .BYTE   0                  ;STATUS BYTE
        .WORD   BUF                ;BUFFER ADDRESS
        .WORD   RLEN               ;RECORD LENGTH(BYTES)
        .WORD   HORN               ;HI ORDER RECORD NUMBER
        .WORD   LORN               ;LO ORDER RECORD NUMBER
        .ENDM
;*************************************************************************
;    THIS MACRO DEFINES BLOCK BLOCKS.
        .MACRO  BLKBLK  CHR,FUNC,BNUM,BUF,LEN
; BLOCK BLOCK DEFINITION
CHR'BKB: .BYTE   FUNC               ;FUNCTION CODE
        .BYTE   0                  ;ERROR STATUS BYTE
CHR'BNM: .WORD   BNUM               ;REQ. BLK NO.
        .WORD   BUF                ;ADDRESS OF BUFFER
        .WORD   LEN                ;LENGTH OF BUFFER
        .ENDM
;*************************************************************************
; THE FOLLOWING MACRO IS USED TO PARTITION FREE CORE
;
        .MACRO  CORD2  ADDR1,ADDR2,NOTLOW
        .GLOBL  FRECOR,CSIZE
        EQUATE
        .NLIST  CND
        .IF     B NOTLOW
        MOV     #FRECOR,R4
        MOV     #CSIZE,R3          ;1/2 OF FREE CORE IN BYTES
        .ENDC
        .LIST   CND
        BIC     #1,R3              ;FORCE TO WD. BOUNDARY
        MOV     R4,ADDR1
        MOV     R3,ADDR1+SIZ-BUF
        ADD     R3,R4
        MOV     R4,ADDR2
        MOV     R3,ADDR2+SIZ-BUF
        .ENDM   CORD2
```

```
;*************************************************************************
;    THIS MACRO DEFINES SEVERAL IPS PARAMETERS.
        .MACRO  EQUATE
; THE FOLLOWING EQUATES PERTAIN TO FILE OPERATIONS
        CRE=0    ;CREATE FUNCTION
        RET=1    ;RETRIEVE FUNCTION
;   THE FOLLOWING EQUATES PERTAIN TO ACCESS TYPE
        RO=1     ;READ ONLY
        WR=2     ;WRITE ONLY
        MO=3     ;MODIFY (READ/WRITE)
;   THE FOLLOWING EQUATES PERTAIN TO FILE TYPE
        CONTIG=0                 ;CONTIGOUOUS FILE
        LINKD=1 ;LINKED FILE
;   THE FOLLOWING EQUATES PERTAIN TO RECORD TYPE
        FLEN=0   ;FIXED LENGTH RECORDS
        VLEN=1   ;VARIABLE LENGTH RECORDS
;   THE FOLLOWING DEFINES CARRAIGE RETURN AND LINE FEED MNEMONICS
        CR=15
        LF=12
;   THE FOLLOWING EQUATES ARE USED TO DEFINE OFFSETS IN THE REQ BLK
        CHR=0    ;CHAR STRING ADR
        NAM=2    ;FILE NAME
        EXT=6    ;FILE EXTENSION
        DLV=10   ;DEVICE
        UNI=12   ;UNIT NO.
        RTP=13   ;ACCESS TYPE
        BUF=14   ;BUFFER ADDRESS
        SIZ=16   ;BUFFER SIZE
        CNT=20   ;NOU OF RECORDS
        LEN=22   ;RECORD LENGTH
        CON=24   ;NO. OF CONTIG REC REQUESTED
        REC=26   ;REC. NO. REQUESTED
;
; THE FOLLWOING EQUATE IS USED TO DEFINE THE
;MAXIMUM NUMBER OF LINES TO BE OUTPUT BEFORE
;REBUILDING THE DISPLAY
;
        LINMAX=50.
;
;THE FOLLOWING IS THE SIZE OF THE HEADER CREATED BY FILEOP IN THE
;USER SUPPLIED BUFFER
        HDRSIZ=62
        .ENDM   EQUATE
```

```
;*****************************************************************
; THE FOLLOWING MACRO IS USED TO MOVE A SPECIFIED
; NUMBER OF BYTES FORM ONE LOCATION TO ANOTHER
; USING THE REGISTERS SPECIFIED
;
        .MACRO  BYTMOV          FROM,REG1,TO,REG2,CNT,REG3
        .ENABL  LSB
        MOV     #FROM,REG1
        MOV     #TO,REG2
        MOV     #CNT,REG3
1$:     MOVB    (REG1)+,(REG2)+
        SUB     REG3,1$
        .DSABL  LSB
        .ENDM   BYTMOV
;*****************************************************************
;THE FOLLOWING MACRO MOVES TEXT INTO FILE HEADERS.
;
        .MACRO  HDRTXT,ADR1,ADR2,ADR3
        ADD     #384.,R0        ;ADR IN HEADER
        MOV     ADR1,R1         ;ADR OF TEXT
        MOVB    (R1)+,(R0)+     ;STORE IN HEADER
        BNE     .-2             ;HANG TILL NULL
        .IF     NB,ADR2
        .IFT
        DEC     R0              ;BACK TO NULL
        MOV     ADR2,R1         ;NEXT SECTION OF TEXT
        MOVB    (R1)+,(R0)+     ;STORE IN HEADER
        BNE     .-2             ;HANG TILL NULL
        .IFF
        .MEXIT
        .ENDC
        .IF     NB,ADR3
        DEC     R0              ;BACK TO NULL
        MOV     ADR3,R1         ;NEXT SECTION OF TEXT
        MOVB    (R1)+,(R0)+     ;STORE IN HEADER
        BNE     .-2             ;HANG TILL NULL
        .ENDC
        .ENDM   HDRTXT
```

```
;*****************************************************************
;    THE FOLLOWING MACRO PUSHES UP TO SIX ITEMS ON THE STACK.
        .MACRO  PUSH,A,B,C,D,E,F
        .IF     NB,A
        MOV     A,-(SP)             ;PUSH A ON STACK
        .ENDC
        .IF     NB,B
        MOV     B,-(SP)             ;PUSH B TO STACK
        .ENDC
        .IF     NB,C
        MOV     C,-(SP)             ;PUSH C TO STACK
        .ENDC
        .IF     NB,D
        MOV     D,-(SP)             ;PUSH D TO STACK
        .ENDC
        .IF     NB,E
        MOV     E,-(SP)             ;PUSH E TO STACK
        .ENDC
        .IF     NB,F
        MOV     F,-(SP)             ;PUSH F TO STACK
        .ENDC
        .ENDM
;*****************************************************************
;    THE FOLLOWING MACRO POPS UP TO SIX ITEMS FROM THE STACK.
        .MACRO  POP,A,B,C,D,E,F
        .IF     NB,A
        MOV     (SP)+,A             ;POP A FROM STACK
        .ENDC
        .IF     NB,B
        MOV     (SP)+,B             ;POP B FROM STACK
        .ENDC
        .IF     NB,C
        MOV     (SP)+,C             ;POP C FROM STACK
        .ENDC
        .IF     NB,D
        MOV     (SP)+,D             ;POP D FROM STACK
        .ENDC
        .IF     NB,E
        MOV     (SP)+,E             ;POP E FROM STACK
        .ENDC
        .IF     NB,F
        MOV     (SP)+,F             ;POP F FROM STACK
        .ENDC
        .ENDM
```

```
;*****************************************************
;    THE MACRO LOAD GENERATES CODE TO LOAD AN OVERLAY.
;      NUM=THREE CHR. ASCII STRING DESCRIBING OVERLAY NO.
;      ENT=ENTRY POINT NO. (ANY ADDRESSING MODE IS LEGAL)
;
        .MACRO  LOAD NUM,ENT
        SYSTEM                          ;DEFINE SYSTEM PARAMS.
        MOV     #RUNFIL,R2              ;ADR. OF RUN FILENAME BLK.
        MOV     #OVRNAM,(R2)+           ;RAD50 PACK OF "OVR"
        MOV     (PC)+,(R2)+             ;RAD50 PACK OF OVERLAY NO.
        .RAD50  /NUM/
        MOV     (PC)+,(R2)+             ;EXTENSION
        .RAD50  /LDA/
        MOV     #IPSUID,@R2             ;USER ID NO.
        MOVB    #UNITNO,RUNLNK+5                ;UNIT NO.
        MOV     #SYDEV,RUNLNK+6         ;DEVICE NAME
        MOV     ENT,ENTRY               ;ENTRY POINT NO.
        JMP     LOADER                  ;GO LOAD IT
        .ENDM   LOAD
;*****************************************************
        .LIST
```

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	*siemens*	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

*U.S. GOVERNMENT PRINTING OFFICE: 1977-714-025